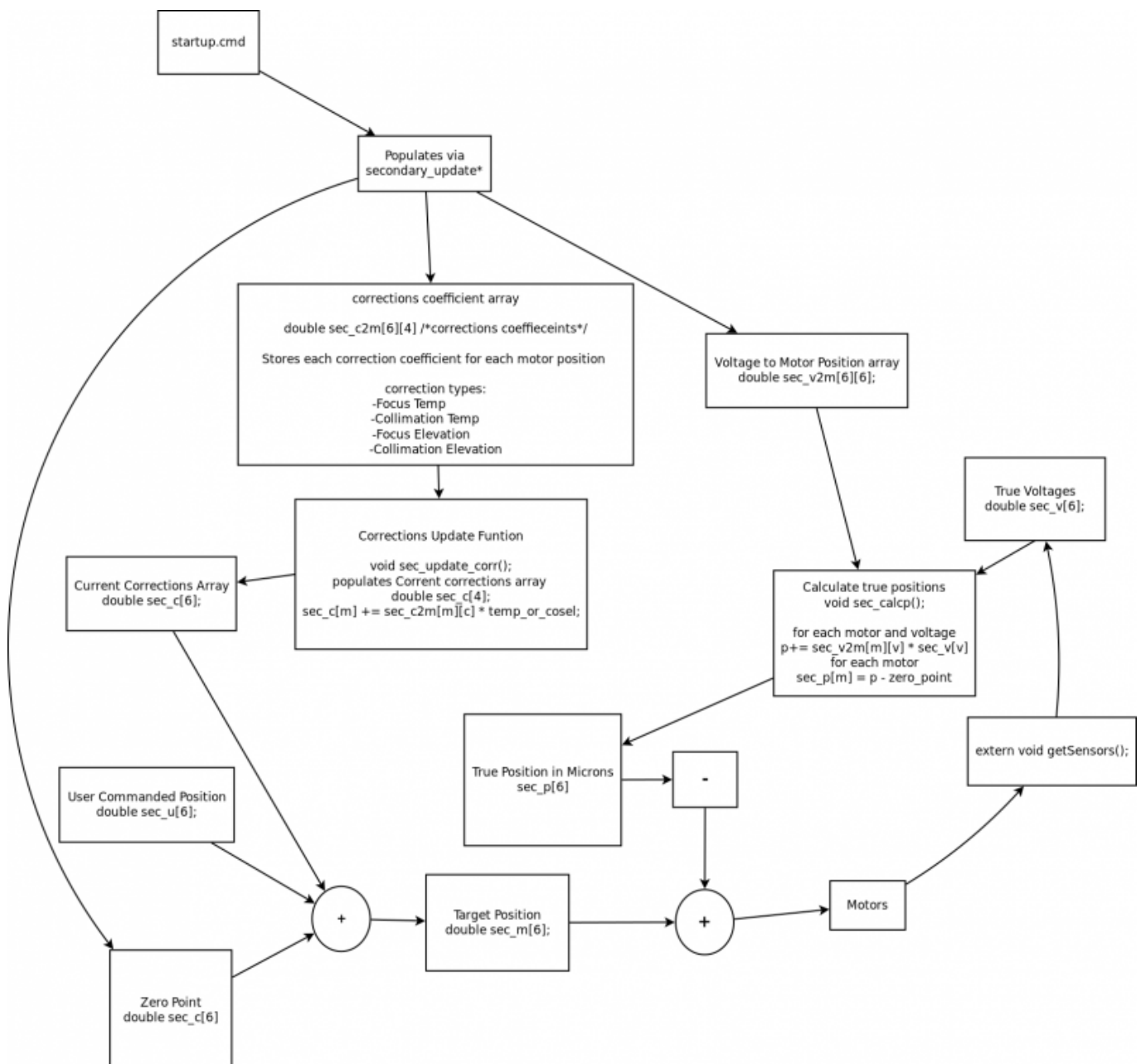


VATT Pseudo Hexapod Legacy

The VATT secondary is focused and collimated with a Pseudo Hexapod. Motion between the axes are coupled but as little as possible and we get uncoupled positions with some linear algebra and iterations. The pseudo hexapod is to be retired for a PI hexapod which will be installed summer 2017

Software

The VATT secondary is controlled using a c program that runs on vatttel. You can find the source in our mogit repository. The most important source file is secmove.c, which controls the autofocus, autocollimation and closed loop motion. A block diagram and the source code for this module are below.



```
/* VATT secondary move routine
 * The six user motions are all cross coupled
 * into the six motors on the VATT secondary.
 * Hence, all the matrix math.
 * This will be a replacement for winsec.c
 */
#define abs(a)                ((a) < 0 ? -(a) : (a))

extern void    sprintf();
extern void    printf();
extern int     sec_checkVoltsOK();
extern void    movev();
extern void    vme8_kill();

double sec_dv = 0.03;        /* maximum allowable voltage difference */
double sec_dvmax = 0.2;      /* maximum before kill */
double sec_min_temp = -40.0; /* minimum sane temperature */
double sec_max_temp = 50.0;  /* maximum sane temperature */
double sec_min_cosel = 0.0;   /* minimum sane cosine(elevation) */
double sec_max_cosel = 1.0;   /* maximum sane cosine(elevation) */
double sec_lastgood_temp;     /* last sane temperature */
double sec_lastgood_cosel;    /* last sane cosine(elevation) */
int     sec_max_mloops = 5;    /* maximum motor loops */

#define V      6      /* number of motors/sensors */
#define M      6      /* number of user motions */
#define TIPX   0      /* arcseconds */
#define TIPY   1
#define TIPZ   2
#define DECENX 3      /* microns */
#define DECENY 4
#define FOCUS  5

double sec_threshold[M] = {2.5, 2.5, 2.5, 4.5, 4.5, 1.0};
char   sec_errormessage[100]; /* error message */
double sec_u0[M];             /* previous user values */
double sec_u[M];              /* user values */
double sec_c[M];              /* corrections */
double sec_m[M];              /* target motions */
double sec_t[V];              /* target voltages */
double sec_v[V];              /* true voltages */
double sec_p[M];              /* true positions */

/* The immediately above positions are derived from the lvdv readings
 * and multiplied by v2m, then the zero offset is removed.
 */

#define C      4      /* number of corrections */
#define FTC    0
```

```

#define FLC      1
#define CTC      2
#define CLC      3

/* the original Steve West matrix
*/
double sec_m2v_sw[V][M] = {
{0.0029680, 0.0051070, -0.0001555, -0.0000931, -0.0001510, -0.0069780},
{0.0016000, -0.0007957, -0.0033810, -0.0033320, -0.0058000, -0.0003111},
{0.0029300, -0.0050660, -0.0000175, -0.0000407, 0.0000389, -0.0069420},
{-0.001475, -0.0010590, -0.0032270, -0.0031640, 0.0057680, -0.0004334},
{-0.005346, -0.0000617, 0.0000176, 0.0000674, 0.0001066, -0.0063050},
{-0.0001785, 0.0017840, -0.0034270, 0.0065690, 0.0000873, -0.0002048}
};

/* All new values per cromwell 10/6/97
* Most of these are now updated in startup.cmd
*/

double sec_z[M] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
double sec_z0[M] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

double sec_m2v[V][M] = {
{0.0029680, 0.0051070, -0.0001555, -0.0000931, -0.0001510, -0.0069780},
{0.0016000, -0.0007957, -0.0033810, -0.0033320, -0.0058000, -0.0003111},
{0.0029300, -0.0050660, -0.0000175, -0.0000407, 0.0000389, -0.0069420},
{-0.001475, -0.0010590, -0.0032270, -0.0031640, 0.0057680, -0.0004334},
{-0.005346, -0.0000617, 0.0000176, 0.0000674, 0.0001066, -0.0063050},
{-0.0001785, 0.0017840, -0.0034270, 0.0065690, 0.0000873, -0.0002048}
};

double sec_v2m[M][V] = {
{ 55.4830132, -3.6541212, 58.2489128, -0.3046370, -125.3519897,
0.4331740, },
{ 98.1537399, -3.2015131, -98.2364426, 0.0305685, -0.2867458,
-0.8237946, },
{ 3.4616468, -100.5291290, 5.1929183, -99.6347046, 5.4746866,
-98.9561157, },
{ -25.0618916, -50.5131302, 29.3149300, -53.1427689, -1.6702332,
100.8553619, },
{ 16.7931938, -85.4085464, 12.2465286, 88.4728470, -33.9353485,
-0.0462751, },
{ -47.9787979, 0.8650305, -47.8929024, 0.9076155, -52.8920479,
0.4418988, },
};

double sec_c2m[M][C] = {
{ 0.0, 0.0, 0.0, 0.0, },
{ 0.0, 0.0, 0.0, 0.0, },
{ 0.0, 0.0, 0.0, 0.0, },

```

```
{ 0.0, 0.0, 0.0, 0.0, },  
{ 0.0, 0.0, 0.0, 0.0, },  
{ 0.0, 0.0, 0.0, 0.0, },  
};
```

```
void  
sec_update_Z (m0, m1, m2, m3, m4, m5)  
double m0;  
double m1;  
double m2;  
double m3;  
double m4;  
double m5;  
{  
    int m;  
    double mm[M];  
  
    mm[0] = m0;  
    mm[1] = m1;  
    mm[2] = m2;  
    mm[3] = m3;  
    mm[4] = m4;  
    mm[5] = m5;  
  
    for (m = 0; m < M; m++) {  
        sec_z[m] = mm[m];  
        sec_z0[m] = mm[m];  
        printf ("%0.7f\n", mm[m]);  
    }  
}
```

```
void  
sec_update_nelsonCT (m0, m1, m2, m3, m4, m5)  
double m0;  
double m1;  
double m2;  
double m3;  
double m4;  
double m5;  
{  
    int m;  
    double mm[M];  
  
    mm[0] = m0;  
    mm[1] = m1;  
    mm[2] = m2;  
    mm[3] = m3;  
    mm[4] = m4;  
    mm[5] = m5;
```

```
    for (m = 0; m < M; m++) {
        sec_c2m[m][CTC] = mm[m];
        printf ("%0.7f\n", mm[m]);
    }
}

void
sec_update_nelsonCL (m0, m1, m2, m3, m4, m5)
double m0;
double m1;
double m2;
double m3;
double m4;
double m5;
{
    int    m;
    double mm[M];

    mm[0] = m0;
    mm[1] = m1;
    mm[2] = m2;
    mm[3] = m3;
    mm[4] = m4;
    mm[5] = m5;

    for (m = 0; m < M; m++) {
        sec_c2m[m][CLC] = mm[m];
        printf ("%0.7f\n", mm[m]);
    }
}

void
sec_update_nelsonFT (m5)
double m5;
{
    sec_c2m[FOCUS][FTC] = m5;
}

void
sec_update_nelsonFL (m5)
double m5;
{
    sec_c2m[FOCUS][FLC] = m5;
}

void
sec_engzero ()
{
    int    m;
```

```
    for (m = 0; m < M; m++) {
        sec_u[m] -= sec_z0[m]-sec_z[m];
        sec_z[m] += sec_z0[m]-sec_z[m];
    }
}

void
sec_userzero ()
{
    int    m;

    for (m = 0; m < M; m++) {
        sec_z[m] += sec_u[m];
        sec_u[m] = 0.0;
    }
}

void
sec_logger (orig, mess)
char    *orig;
char    *mess;
{
    void    vxcall();

    vxcall ("_logger_it", orig, mess, 0,0,0,0,0,0,0,0);
}

int
sec_telslewing()
{
    int    telslewing = 0;
    void    vxcall();

    vxcall ("_telx_sec_telslewing", &telslewing, 0,0,0,0,0,0,0,0);
    return (telslewing);
}

int    sec_working;
double sec_azimuth;
double sec_derotator;
double sec_elevation;    /* current altitude in radians */
double sec_temperature;    /* current temperature in degrees C */
static int    sec_autofocus;
static int    sec_autocoll;

void
sec_update_tl ()    /* update temperature and elevation */
{
```

```
void    vxcall();

vxcall ("_telx_sec_temperature", &sec_temperature, 0,0,0,0,0,0,0,0,0,0);
vxcall ("_telx_sec_elevation"   , &sec_elevation   , 0,0,0,0,0,0,0,0,0,0);
vxcall ("_telx_sec_azimuth"    , &sec_azimuth    , 0,0,0,0,0,0,0,0,0,0);
vxcall ("_telx_sec_derotator"   , &sec_derotator   , 0,0,0,0,0,0,0,0,0,0);
}

void
sec_autoreset ()
{
}

void
sec Autofocus_set (status)
int    status;
{
    sec Autofocus = status;
}

int
sec Autofocus_get ()
{
    return (sec Autofocus);
}

void
sec Autocoll_set (status)
int    status;
{
    sec Autocoll = status;
}

int
sec Autocoll_get ()
{
    return (sec Autocoll);
}

void
sec_update_corr ()
{
    int    m, c;
    double sc;
    double temp, cosel;
    double d[C];
    double cos();

    sec_update_tl ();
    temp = sec_temperature;
    cosel = cos(sec_elevation);
```

```
/* sanity checks
*/
if (sec_min_temp < temp && temp < sec_max_temp) {
    sec_lastgood_temp = temp;
} else {
    temp = sec_lastgood_temp;
}
if (sec_min_cosel < cosel && cosel < sec_max_cosel) {
    sec_lastgood_cosel = cosel;
} else {
    cosel = sec_lastgood_cosel;
}

d[FTC] = sec_autofocus ? temp : 0.0;
d[FLC] = sec_autofocus ? cosel : 0.0;
d[CTC] = sec_autocoll ? temp : 0.0;
d[CLC] = sec_autocoll ? cosel : 0.0;

for (m = 0; m < M; m++) {
    sc = 0.0;
    for (c = 0; c < C; c++) {
        sc += sec_c2m[m][c] * d[c];
    }
    sec_c[m] = sc;
}

}

void
sec_calct ()    /* calculate targets */
{
    int    m, v;
    double t;

    for (v = 0; v < V; v++) {
        t = 0.0;
        for (m = 0; m < M; m++) {
            sec_m[m] = sec_u[m]+sec_c[m]+sec_z[m];
            t += sec_m2v[v][m] * sec_m[m];
        }
        sec_t[v] = t;
    }
}

void
sec_calcp ()    /* calculate true positions */
{
    int    m, v;
    double p;
```

```

    for (m = 0; m < M; m++) {
        p = 0.0;
        for (v = 0; v < V; v++) {
            p += sec_v2m[m][v] * sec_v[v];
        }
        sec_p[m] = p - sec_z[m];
    }
}

int
sec_checkMove()
{
    int    v;
    int    mstatus = 0;
    int    rstatus = 0;
    double residual;
    char    errormessage[100];
    int    strlen();

    errormessage[0] = 0;
    for (v = 0; v < V; v++) {
        residual = sec_t[v]-sec_v[v];
        sprintf (errormessage+strlen(errormessage), " %.4f", residual);
        if (abs(residual) > sec_dv)
            mstatus = 1;
        if (abs(residual) > sec_dvmax)
            rstatus = 1;
    }
    if (mstatus)
        sprintf (sec_errormessage, "residuals =%s", errormessage);
    return (rstatus);
}

void
sec_voltsNotOK ()
{
    int    v;
    char    errormessage[100];
    int    strlen();

    errormessage[0] = 0;
    for (v = 0; v < V; v++) {
        sprintf (errormessage+strlen(errormessage), " %.4f", sec_t[v]);
    }
    sprintf (sec_errormessage, "bad target =%s", errormessage);
}

void
sec_sprintf (c, array)
char    *c;

```

```
double array[];
{
    int    m;
    int    strlen();

    *c = 0;
    for (m = 0; m < M; m++) {
        sprintf (c+strlen(c), " %12.6f", array[m]);
    }
}

void
sec_sprintfu (c, array, array0)
char    *c;
double array[];
double array0[];
{
    int    m;
    int    strlen();

    *c = 0;
    for (m = 0; m < M; m++) {
        sprintf (c+strlen(c), " %d", array[m] != array0[m]);
        array0[m] = array[m];
    }
}

void
sec_log_servc ()
{
    char    c[100];
    double    readmotorcurrent ();

    sprintf (c, "%12.6f %12.6f %12.6f", readmotorcurrent (4),
        readmotorcurrent (2), readmotorcurrent (3));
    sec_logger ("serv_c", c);
}

void
sec_log ()
{
    char    c[100];
    int    strlen();
    double cos();

    sprintf (c, "%12.6f %12.6f %12.6f %12.6f %12.6f",
        sec_elevation*57.29577951,
        cos(sec_elevation),
        sec_temperature,
```

```

        sec_azimuth*57.29577951,
        sec_derotator*57.2957795);
sec_logger ("sec_el", c);
sec_sprintf (c, sec_u);
sec_sprintfu (c+strlen(c), sec_u, sec_u0);
sec_logger ("sec_u", c);
sec_sprintf (c, sec_c);
sec_logger ("sec_c", c);
sec_sprintf (c, sec_z);
sec_logger ("sec_z", c);
sec_sprintf (c, sec_m);
sec_logger ("sec_m", c);
sec_sprintf (c, sec_t);
sec_logger ("sec_t", c);
sec_sprintf (c, sec_v);
sec_logger ("sec_v", c);
sec_sprintf (c, sec_p);
sec_logger ("sec_p", c);
sec_log_servc ();
}

void
sec_loop_init ()
{
    void    getSensors();
    int     m;

    getSensors ();
    sec_calcp ();
    for (m = 0; m < M; m++) {
        sec_u[m] = sec_p[m];
        sec_u0[m] = sec_u[m];
    }
    sec_log ();
}

void
sec_loop (timeout)
int     timeout;    /* !0 if triggered by timer */
{
    int     n;
    int     m;
    int     big;
    void     strcpy();
    void     getSensors();

    if (!timeout || ((sec_autofocus || sec_autocoll) && !sec_telslewing())) {
        sec_working = 1;
        sec_update_corr ();
        sec_calct ();
        if (sec_checkVoltsOK (sec_t)) {

```

```
    getSensors ();
    sec_calcp ();
    for (n = 0; n < sec_max_mloops; n++) {
        big = 0;
        for (m = 0; m < M; m++) {
            big |= abs(sec_m[m]-sec_z[m]-sec_p[m])
                > sec_threshold[m];
        }
        if (!big)
            break;
        strcpy (sec_errormessage, "moving");
        movev (sec_t);
        strcpy (sec_errormessage, "");
        getSensors ();
        sec_calcp ();
    }
    if (sec_checkMove ()) {
        vme8_kill ();
    }
} else {
    sec_voltsNotOK ();
    sec_calcp ();
}
sec_log ();
sec_working = 0;
}
}
```

```
void
sec_printf (id, array)
char *id;
double array[];
{
    int m;

    printf ("%s:", id);
    for (m = 0; m < M; m++) {
        printf (" %12.6f", array[m]);
    }
    printf ("\n");
}
```

```
void
sec_printd (id, array)
char *id;
int array[];
{
    int m;
```

```

    printf ("%s:", id);
    for (m = 0; m < M; m++) {
        printf (" %12d", array[m]);
    }
    printf ("\n");
}

extern int    sec_s[];

void
sec_print ()
{
    printf ("%s\n", sec_errormessage);
    sec_printf ("z", sec_z);
    sec_printf ("u", sec_u);
    sec_printf ("c", sec_c);
    sec_printf ("m", sec_m);
    sec_printf ("t", sec_t);
    sec_printf ("s", sec_s);
    sec_printf ("v", sec_v);
    sec_printf ("p", sec_p);
    sec_errormessage[0] = 0;
}

void
telx_log_servc ()
{
    sec_log_servc ();
}

void
telx_lvdts ()
{
    int    v;
    char    errormessage[100];
    int    strlen();

    errormessage[0] = 0;
    for (v = 0; v < V; v++) {
        sprintf (errormessage+strlen(errormessage), " %.4f", sec_v[v]);
    }
    sprintf (sec_errormessage, "lvdts =%s", errormessage);
}

void
telx_rlvdts ()
{
    void    getSensors();

    getSensors ();
    telx_lvdts ();
}

```

```
    sec_calcp ();
    sec_log ();
}

void
plvdtls ()
{
    void    getSensors();
    int     v;

    getSensors ();
    telx_lvdtls ();
    sec_calcp ();

    printf ("lvdtls =");
    for (v = 0; v < V; v++) {
        printf (" %.4f", sec_v[v]);
    }
    printf ("\n");
}

void
pmcurr ()
{
    double   readmotorcurrent ();

    printf ("%12.6f %12.6f %12.6f\n", readmotorcurrent (4),
        readmotorcurrent (2), readmotorcurrent (3));
}

int     telvatt_telfocustimeout = 70;
int     telvatt_telfocuscmdlevel;

void
telvatt_telfocus (c)
char    *c;
{
    void     sleep();
    void     secondary_trigger();
    int      sscanf();
    double   focus;
    int      timeout = telvatt_telfocustimeout;

    if ((telvatt_telfocuscmdlevel > 0) && (sscanf(c, "%lf",&focus) == 1)) {
        sec_u[FOCUS] = focus;
        sec_working = 1;
        secondary_trigger();
        while (sec_working && timeout--)
            sleep (1);
    }
}
```

```

    }
    sprintf (c, "%.2f", sec_u[FOCUS]);
}
#ifdef OUTOFDATE
/* This is an arbitrary zero point for the user motions.
 *
 * It was derived using Tom T.'s goodplace lvdv values
 * and multiplying by the v2m matrix.
 * { 4.348, -6.021, 5.151, -5.925, 3.126, -5.148 }
double sec_z[M] = {97.0968554, -46.2452714, 1763.9614446,
                  136.6180699, 20.2980418, -633.5096204};
 * new values 3/25/97 via cromwell->nelson->harvey
double sec_z[M] = {-103.1331446, 38.2247286, 1763.9614446,
                  150.2880699, -0.5719582, -658.7596204};
double sec_z0[M] = {-103.1331446, 38.2247286, 1763.9614446,
                  150.2880699, -0.5719582, -658.7596204};
 * new values 5/30/97
 */
double sec_z[M] = {-273.5316558, 168.0022732, 1962.0991461,
                  26.3139608, -168.5847471, -427.0699187};
double sec_z0[M] = {-273.5316558, 168.0022732, 1962.0991461,
                  26.3139608, -168.5847471, -427.0699187};

/* Steve West's original matrix
{0.0029680, 0.0051070, -0.0001555, -0.0000931, -0.0001510, -0.0069780},
{0.0016000, -0.0007957, -0.0033810, -0.0033320, -0.0058000, -0.0003111},
{0.0029300, -0.0050660, -0.0000175, -0.0000407, 0.00000389, -0.0069420},
{-0.001475, -0.0010590, -0.0032270, -0.0031640, 0.0057680, -0.0004334},
{-0.005346, -0.0000617, 0.0000176, 0.0000674, 0.0001066, -0.0063050},
{-0.0001785, 0.0017840, -0.0034270, 0.0065690, 0.0000873, -0.0002048}
 */

/* The following matrix of user motions (across) to lvdv voltages (down)
 * already has the following correction applied.
 */

/* There is correction in dtx and dty when applying focus. There are
 * 2 reasons: the positioner is mounted slightly tilted wrt to
 * the primary axis, and the primary comes up on its supports
 * slightly differently each time causing a small but variable
 * tilt term. For now, this correction approximates the constant
 * term for mounting misalignment and the tilt for which the primary
 supports
 * usually produce.
 * dtxSlope = -0.1166667;
 * dtySlope = 0.0166667;
 */

double sec_m2v[V][M] = {
{ 0.0029680, 0.0051070, -0.0001555, -0.0000931, -0.0001510, -0.0072391, },
{ 0.0016000, -0.0007957, -0.0033810, -0.0033320, -0.0058000, -0.0005110, },

```

```
{ 0.0029300, -0.0050660, -0.0000175, -0.0000407, 0.0000389, -0.0073683, },
{-0.0014750, -0.0010590, -0.0032270, -0.0031640, 0.0057680, -0.0002790, },
{-0.0053460, -0.0000617, 0.0000176, 0.0000674, 0.0001066, -0.0056823, },
{-0.0001785, 0.0017840, -0.0034270, 0.0065690, 0.0000873, -0.0001542, },
};

/* The following matrix of lvdt voltages (across) to user motions (down)
 * is the inversion of the above.
 */

double sec_v2m[M][V] = {
{ 49.8854752, -3.5532007, 52.6613884, -0.1987483, -131.5227203, 0.4847288,
},
{ 98.9533844, -3.2159302, -97.4382248, 0.0154418, 0.5947783, -0.8311595,
},
{ 3.4616475, -100.5291367, 5.1929212, -99.6347046, 5.4746842, -98.9561081,
},
{-25.0618954, -50.5131378, 29.3149300, -53.1427727, -1.6702307, 100.8553619,
},
{ 16.7931862, -85.4085312, 12.2465229, 88.4728546, -33.9353409, -0.0462707,
},
{-47.9788094, 0.8650309, -47.8929062, 0.9076155, -52.8920441, 0.4418987,
},
};

/* The following matrix of (T,L) corrections (across)
 * {focus_delta_temperature, focus_delta_sin_elevation,
 * collimation_delta_temperature, collimation_delta_sin_elevation}
 * to user motion corrections (down)
 *
 * These were derived from Cromwell's original (June 10, 1996) coefficients
 * for ELCdv and TCdv by multiplying by v2m.
 * 0.2230000 0.0914000 -0.1024000 -0.0426000 0.0031000
0.1702000
 * 0.2148000 0.5129000 0.0363000 0.1717000 -0.0712000
0.4409000
 */

double sec_c2m[M][C] = {
{ 0.0, 0.0, 20.3485814, 5.0904193, },
{ 0.0, 0.0, 15.6625737, 31.6100656, },
{ 0.0, 0.0, -111.7561536, -21.5290905, },
{ 0.0, 0.0, 5.2340839, 6.2167347, },
{ 0.0, 0.0, -22.1677258, -9.1975216, },
{ 15.0, -120.0, -7.4841021, -5.8433957, },
};

/* values as of 10/6/97
 */
```

```

double sec_c2m[M][C] = {
{ 0.0, 0.0, 21.2217299, 5.7721492, },
{ 0.0, 0.0, 15.5378383, 31.5126765, },
{ 0.0, 0.0, -111.7561535, -21.5290909, },
{ 0.0, 0.0, 5.2340894, 6.2167361, },
{ 0.0, 0.0, -22.1677345, -9.1975223, },
{ 15.0, -120.0, -7.4840994, -5.8433935, },
};

void
sec_update_cromwellB (v0, v1, v2, v3, v4, v5)
double v0;
double v1;
double v2;
double v3;
double v4;
double v5;
{
    int v, m;
    double vv[V];
    double mm[M];

    vv[0] = v0;
    vv[1] = v1;
    vv[2] = v2;
    vv[3] = v3;
    vv[4] = v4;
    vv[5] = v5;

    for (m = 0; m < M; m++) {
        mm[m] = 0.0;
        for (v = 0; v < V; v++) {
            mm[m] += sec_v2m[m][v]*vv[v];
        }
        sec_c2m[m][CLC] = mm[m];
        printf ("%0.7f\n", mm[m]);
    }
}

void
sec_update_cromwellB0 ()
{
    sec_update_cromwellB (0.2230000, 0.0914000, -0.1024000,
        -0.0426000, 0.0031000, 0.1702000);
}

void
sec_update_cromwellC (v0, v1, v2, v3, v4, v5)
double v0;

```

```
double v1;
double v2;
double v3;
double v4;
double v5;
{
    int    v, m;
    double vv[V];
    double mm[M];

    vv[0] = v0;
    vv[1] = v1;
    vv[2] = v2;
    vv[3] = v3;
    vv[4] = v4;
    vv[5] = v5;

    for (m = 0; m < M; m++) {
        mm[m] = 0.0;
        for (v = 0; v < V; v++) {
            mm[m] += sec_v2m[m][v]*vv[v];
        }
        sec_c2m[m][CTC] = mm[m];
        printf ("%0.7f\n", mm[m]);
    }
}

void
sec_update_cromwellC0 ()
{
    sec_update_cromwellC (0.2148000,0.5129000,0.0363000,
        0.1717000,-0.0712000,0.44090000);
}

void
sec_update_cromwellZ (v0, v1, v2, v3, v4, v5)
double v0;
double v1;
double v2;
double v3;
double v4;
double v5;
{
    int    v, m;
    double vv[V];
    double mm[M];

    vv[0] = v0;
    vv[1] = v1;
```

```

vv[2] = v2;
vv[3] = v3;
vv[4] = v4;
vv[5] = v5;

for (m = 0; m < M; m++) {
    mm[m] = 0.0;
    for (v = 0; v < V; v++) {
        mm[m] += sec_v2m[m][v]*vv[v];
    }
    sec_z[m] = mm[m];
    sec_z0[m] = mm[m];
    printf ("%0.7f\n", mm[m]);
}

void
sec_update_cromwellZ0 ()
{
    sec_update_cromwellZ (0.0,0.0,0.0,
        0.0,0.0,0.0);
}

/* auto variables */
/* focus versus temperature */
double sec_ft0; /* starting point */
double sec_ft1; /* last measurement */
double sec_ft2; /* measurement at last adjustment */
double sec_fdt0; /* accumulated delta */
double sec_fdt; /* delta */
/* focus versus elevation */
double sec_fl0;
double sec_fl1;
double sec_fl2;
double sec_fdsl0;
double sec_fdsl;
/* collimation versus temperature */
double sec_ct0;
double sec_ct1;
double sec_ct2;
double sec_cdt0;
double sec_cdt;
/* collimation versus elevation */
double sec_cl0;
double sec_cl1;
double sec_cl2;
double sec_cdsl0;
double sec_cdsl;
#endif

```

Last
update:
2017/07/11 14:29 vatt:legacy_psuedo-hexpod https://lavinia.as.arizona.edu/~tscopewiki/doku.php?id=vatt:legacy_psuedo-hexpod&rev=1499808554

From:
<https://lavinia.as.arizona.edu/~tscopewiki/> - **MOON**

Permanent link:
https://lavinia.as.arizona.edu/~tscopewiki/doku.php?id=vatt:legacy_psuedo-hexpod&rev=1499808554

Last update: **2017/07/11 14:29**

