Speeds

15 December 2004

TABLE OF CONTENTS

Section **Page**

[1] `$Header: d:/UnitSpeeds/RCS/Speeds.tex,v 1.16 2004-12-15 07:11:08-08 Hamilton Exp Hamilton $`
[2] `tocdepth = 2`

LIST OF FIGURES

LIST OF TABLES

This page intentionally left blank

1 PT motion speeds

At Pelco considerable effort has been made since the middle 1990's to provide excellent control of various integrated positioning systems. Originally all Pan and Tilt systems were of the "fixed speed" verity. (Pelco still manufactures fixed speed units.)

1.1 Pan and Tilt Speeds

1.1.1 Pan Speeds

Each column of this table has been taken from a release of software for Pelco domes (Intercept, Spectra) or pan/tilt units (Esprit). The data was obtained by going through the source code saved in Document Control. All speeds in the table are in degrees per second (^{o}/sec). The bold values in columns **E** and **H** are the current values being used in Spectra III and Esprit units. The column headings represent the following revisions/types of software:

A Intercept `DRD08A12u3_R3.06` Pan Speeds, Figure 1, page 12

B Spectra I `DD5x- PRGSPCTFW106` Pan Speeds, Figure 3, page 17

C Spectra II `PG53-0001-0206` "0206" Pan Speeds"OLD_SPEED_TABLE", Figure 5, page 19

D Spectra II `PG53-0001-0206` "0206" Pan Speeds "non-OLD_SPEED_TABLE", Figure 6, page 21

E Spectra II `PG53-0060-0308` "0308" Pan Speeds, Figure 9, page 25

F Esprit `PG53-0026-0100` Pan Speeds "NOTEST", Figure 13, page 31

G Esprit `PG53-0026-0100` Pan Speeds "not-NOTEST", Figure 14, page 32

H Esprit `PG53-0096-0210` Pan Speeds, Figure 17, page 35

[3] `$Header: d:/UnitSpeeds/RCS/Speeds.inc,v 1.9 2004-11-10 09:42:26-08 Hamilton Exp Hamilton $`

Index	A	B	C	D	**E**	F	G	**H**
0	1.5	.6	.6	.5	**.5**	.2	.2	**.2**
1	1.5	.6	.6	.5	**.5**	.2	.2	**.2**
2	1.5	.6	.6	.5	**.5**	.3	.3	**.3**
3	1.5	.6	.6	.5	**.5**	.3	.3	**.3**
4	1.5	.6	.6	.5	**.5**	.4	.4	**.4**
5	1.5	.6	.6	.5	**.5**	.5	.5	**.5**
6	1.5	.6	.6	.5	**.5**	.5	.7	**.7**
7	1.5	.6	.6	.9	**.5**	.9	.9	**.9**
8	1.5	1.3	1.3	1.3	**.5**	1.3	1.2	**1.2**
9	1.5	2.0	2.0	1.6	**.5**	1.6	1.5	**1.5**
10	1.9	2.7	2.7	2.0	**.6**	2.0	1.8	**1.8**
11	3.0	3.4	3.4	2.3	**.7**	2.3	2.1	**2.1**
12	3.2	3.4	4.1	2.7	**.7**	2.7	2.5	**2.5**
13	4.5	3.4	4.8	3.0	**.8**	3.0	2.9	**2.9**
14	4.9	6.3	5.6	3.4	**.9**	3.4	3.3	**3.3**
15	5.4	6.3	6.3	3.7	**1.0**	3.7	3.7	**3.7**
16	5.4	7.0	7.0	4.1	**1.0**	4.1	4.1	**4.1**
17	5.4	7.7	7.7	4.5	**1.1**	4.5	4.5	**4.5**
18	5.4	8.4	8.4	4.8	**1.3**	4.8	4.8	**4.8**
19	9.0	8.4	9.1	5.2	**1.4**	5.2	5.2	**5.2**
20	9.4	8.4	9.8	5.6	**1.5**	5.6	5.6	**5.6**
21	10.0	11.2	10.5	5.9	**1.7**	5.9	5.9	**5.9**
22	10.4	11.2	11.2	6.3	**1.8**	6.3	6.3	**6.3**
23	10.9	11.9	11.9	6.7	**2.0**	6.7	6.7	**6.7**
24	11.4	12.6	12.6	7.1	**2.2**	7.1	7.1	**7.1**
25	11.8	13.3	13.3	7.5	**2.4**	7.5	7.5	**7.5**
26	12.4	14.0	14.0	7.8	**2.6**	7.8	7.8	**7.8**
27	12.8	14.7	14.7	8.2	**2.9**	8.2	8.2	**8.2**
28	13.3	15.4	15.4	8.6	**3.2**	8.6	8.6	**8.6**
29	13.8	16.1	16.1	9.0	**3.5**	9.0	9.0	**9.0**
30	14.3	16.8	16.8	9.4	**3.8**	9.4	9.4	**9.4**
31	16.5	17.5	17.5	9.9	**4.2**	9.9	9.9	**9.9**
32	16.9	18.2	18.2	10.3	**4.6**	10.3	10.3	**10.3**
33	17.4	18.9	18.9	10.7	**5.0**	10.7	10.7	**10.7**
34	17.8	19.6	19.6	11.1	**5.5**	11.1	11.1	**11.1**

Continued on the next page.

Continued from the previous page.

Index	A	B	C	D	**E**	F	G	**H**
35	18.3	20.3	20.3	11.6	**6.0**	11.6	11.6	**11.6**
36	18.7	21.0	21.0	12.1	**6.6**	12.1	12.1	**12.1**
37	19.1	21.7	21.7	12.5	**7.3**	12.5	12.5	**12.5**
38	19.6	22.4	22.4	13.0	**8.0**	13.0	13.0	**13.0**
39	20.0	23.1	23.1	13.5	**8.7**	13.5	13.5	**13.5**
40	20.4	23.8	23.8	14.1	**9.6**	14.1	14.1	**14.1**
41	20.9	24.5	24.5	14.6	**10.5**	14.6	14.6	**14.6**
42	21.3	25.2	25.2	15.2	**11.5**	15.2	15.2	**15.2**
43	21.8	25.9	25.9	15.7	**12.6**	15.7	15.7	**15.7**
44	25.5	26.6	26.6	16.4	**13.9**	16.4	16.4	**16.4**
45	26.1	27.3	27.3	17.0	**15.2**	17.0	17.0	**17.0**
46	26.6	28.0	28.0	17.7	**16.7**	17.7	17.7	**17.7**
47	27.2	28.8	28.8	18.4	**18.3**	18.4	18.4	**18.4**
48	27.7	29.5	29.5	19.1	**20.0**	19.1	19.1	**19.1**
49	28.3	30.2	30.2	19.9	**22.0**	19.9	19.9	**19.9**
50	28.8	30.9	30.9	20.8	**24.1**	20.8	20.8	**20.8**
51	29.4	31.6	31.6	21.7	**26.4**	21.7	21.7	**21.7**
52	29.9	32.3	32.3	22.7	**29.0**	22.7	22.7	**22.7**
53	30.5	33.0	33.0	23.7	**31.8**	23.7	23.7	**23.7**
54	31.0	33.7	33.7	24.8	**34.9**	24.8	24.8	**24.8**
55	31.6	34.4	34.4	26.0	**38.2**	26.0	26.0	**26.0**
56	32.1	35.1	35.1	27.3	**41.9**	27.3	27.3	**27.3**
57	32.7	35.8	35.8	28.7	**46.0**	28.7	28.7	**28.7**
58	33.2	36.5	36.5	30.2	**50.4**	30.2	30.2	**30.2**
59	33.8	37.2	37.2	31.8	**55.3**	31.8	31.8	**31.8**
60	34.3	37.9	37.9	33.6	**60.7**	33.6	33.6	**33.6**
61	34.9	38.6	38.6	35.6	**66.5**	35.6	35.6	**35.6**
62	35.5	39.3	39.3	37.7	**72.9**	37.7	37.7	**37.7**
63	36.0	40.0	40.0	40.0	**80.0**	40.0	40.0	**40.0**

1.1.2 Tilt Speeds

Each column of this table has been taken from a release of software for Pelco domes (Intercept, Spectra) or pan/tilt units (Esprit). The data was obtained by going through the source code saved in Document Control. All speeds in the table are in degrees per second (^{o}/sec). The bold values in columns **E** and **H** are the current values being used in Spectra III and Esprit units. The column headings represent the following revisions/types of software:

A Intercept `DRD08A12u3_R3.06` Tilt Speeds, Figure 2, page 14

B Spectra I `DD5x- PRGSPCTFW106` Tilt Speeds, Figure 4, page 18

C Spectra II `PG53-0001-0206` "0206" Tilt Speeds "non-OLD_SPEED_TABLE", Figure 7, page 23

D Spectra II `PG53-0060-0331` "0331" NTSC Tilt Speeds, Figure 11, page 28

E Spectra II `PG53-0060-0331` "0331" PAL Tilt Speeds, Figure 12, page 29

F Esprit `PG53-0026-0100` Tilt Speeds "OLD_SPEED_TABLE", Figure 15, page 33

G Esprit `PG53-0026-0100` Tilt Speeds "not-OLD_SPEED_TABLE", Figure 16, page 34

H Esprit `PG53-0096-0210` Tilt Speeds, Figure 18, page 36

Index	A	B	C	**D**	**E**	F	G	H
0	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
1	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
2	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
3	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
4	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
5	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
6	1.5	.6	.5	**.5**	**.5**	.5	.5	**.5**
7	1.5	.6	.9	**.9**	**.9**	.9	.7	**.7**
8	1.5	1.3	1.3	**1.3**	**1.3**	1.3	.9	**.9**
9	1.5	2.0	1.6	**1.6**	**1.6**	1.6	1.1	**1.1**
10	2.0	2.7	2.0	**2.0**	**2.0**	2.0	1.3	**1.3**
11	2.5	3.4	2.3	**2.3**	**2.3**	2.3	1.4	**1.4**
12	3.0	4.1	2.7	**2.7**	**2.7**	2.7	1.6	**1.6**
13	3.6	4.9	3.0	**3.0**	**3.0**	3.0	1.8	**1.8**
14	4.1	5.6	3.4	**3.4**	**3.4**	3.4	2.0	**2.0**
15	4.6	6.3	3.7	**3.7**	**3.7**	3.7	2.2	**2.2**
16	5.1	7.0	4.1	**4.1**	**4.1**	4.1	2.3	**2.3**
17	7.7	7.7	4.5	**4.5**	**4.5**	4.5	2.5	**2.5**
18	8.1	8.4	4.8	**4.8**	**4.8**	4.8	2.7	**2.7**
19	8.5	9.1	5.2	**5.2**	**5.2**	5.2	2.9	**2.9**
20	8.9	9.8	5.6	**5.6**	**5.6**	5.6	3.1	**3.1**
21	9.2	10.5	5.9	**5.9**	**5.9**	5.9	3.3	**3.3**
22	9.6	11.2	6.3	**6.3**	**6.3**	6.3	3.5	**3.5**
23	10.0	11.9	6.7	**6.7**	**6.7**	6.7	3.6	**3.6**
24	10.4	12.6	7.1	**7.1**	**7.1**	7.1	3.8	**3.8**
25	10.8	13.3	7.5	**7.5**	**7.5**	7.5	4.0	**4.0**
26	11.2	14.0	7.8	**7.8**	**7.8**	7.8	4.2	**4.2**
27	11.6	14.7	8.2	**8.2**	**8.2**	8.2	4.4	**4.4**
28	15.4	15.4	8.6	**8.6**	**8.6**	8.6	4.6	**4.6**
29	15.9	16.1	9.0	**9.0**	**9.0**	9.0	4.9	**4.9**
30	16.3	16.8	9.4	**9.4**	**9.4**	9.4	5.1	**5.1**
31	16.8	17.5	9.9	**9.9**	**9.9**	9.9	5.3	**5.3**
32	17.3	18.2	10.3	**10.3**	**10.3**	10.3	5.5	**5.5**
33	17.8	18.9	10.7	**10.7**	**10.6**	10.7	5.7	**5.7**
34	18.2	19.6	11.1	**11.1**	**10.7**	11.1	6.0	**6.0**

Continued on the next page.

Continued from the previous page.

Index	A	B	C	**D**	**E**	F	G	H
35	18.7	20.3	11.6	**11.6**	**11.8**	11.6	6.2	**6.2**
36	19.2	21.0	12.1	**12.1**	**12.1**	12.1	6.4	**6.4**
37	19.7	21.7	12.5	**12.5**	**12.5**	12.5	6.7	**6.7**
38	20.1	22.4	13.0	**12.8**	**13.0**	13.0	6.9	**6.9**
39	20.6	23.1	13.5	**13.1**	**13.5**	13.5	7.2	**7.2**
40	21.1	23.8	14.1	**13.9**	**14.1**	14.1	7.5	**7.5**
41	21.5	24.5	14.6	**14.6**	**14.6**	14.6	7.8	**7.8**
42	22.0	25.2	15.2	**15.2**	**15.2**	15.2	8.1	**8.1**
43	22.5	25.9	15.7	**15.7**	**15.7**	15.7	8.4	**8.4**
44	23.0	26.6	16.4	**16.4**	**16.4**	16.4	8.7	**8.7**
45	23.4	27.3	17.0	**17.0**	**17.0**	17.0	9.0	**9.0**
46	23.9	28.0	17.7	**17.7**	**17.7**	17.7	9.4	**9.4**
47	24.4	28.8	18.4	**18.4**	**18.4**	18.4	9.7	**9.7**
48	28.3	29.5	19.1	**19.1**	**19.1**	19.1	10.1	**10.1**
49	28.8	30.2	19.9	**19.9**	**19.9**	19.9	10.5	**10.5**
50	29.3	30.9	20.8	**20.8**	**20.4**	20.8	11.0	**11.0**
51	29.8	31.6	21.7	**21.7**	**20.6**	21.7	11.4	**11.4**
52	30.4	32.3	22.7	**22.7**	**24.9**	22.7	11.9	**11.9**
53	30.9	33.0	23.7	**23.7**	**25.2**	23.7	12.4	**12.4**
54	31.4	33.7	24.8	**24.8**	**25.3**	24.8	13.0	**13.0**
55	31.9	34.4	26.0	**25.3**	**25.5**	26.0	13.5	**13.5**
56	32.4	35.1	27.3	**29.0**	**26.0**	27.3	14.2	**14.2**
57	32.9	35.8	28.7	**30.0**	**26.5**	28.7	14.8	**14.8**
58	33.4	36.5	30.2	**31.0**	**27.0**	30.2	15.5	**15.5**
59	34.0	37.2	31.8	**32.0**	**38.0**	31.8	16.3	**16.3**
60	34.5	37.9	33.6	**33.6**	**29.0**	33.6	17.1	**17.1**
61	35.0	38.6	35.6	**35.6**	**37.6**	35.6	18.0	**18.0**
62	35.5	39.3	37.7	**37.0**	**40.0**	37.7	19.0	**19.0**
63	36.0	40.0	40.0	**44.0**	**44.0**	40.0	20.0	**20.0**

1.2 Intercept domes

With the Intercept line of equipment some thought was applied to make the units as acoustically quiet and controllable as possible. The work done on the Intercept products resulted in the pan and tilt speed tables shown in Figure 1, page 12 and Figure 2, page 14, respectively. These two graphs show that the speeds were generally linear with increasing speed commands. There are some speeds that were delibertly "skipped over" to avoid mechanical resonances in the units. The Intercept systems consisted of many models in several basic styles. The styles were:

1. 8 and 14 inch diameter domes.
2. Fixed and variable speed units.
3. With and without preset capability.
4. D, P and Coaxitron Protocol compatibility.

An internal description of the workings of the Intercept is shown in Section 4, page 46. The Intercept line of domes is no longer being made. However many are still in use. The calculations related to various Intercept speeds are shown in Section 5, page 70.

1.2.1 Intercept, version `DRD08A12u3_R3.06`, `data.c`

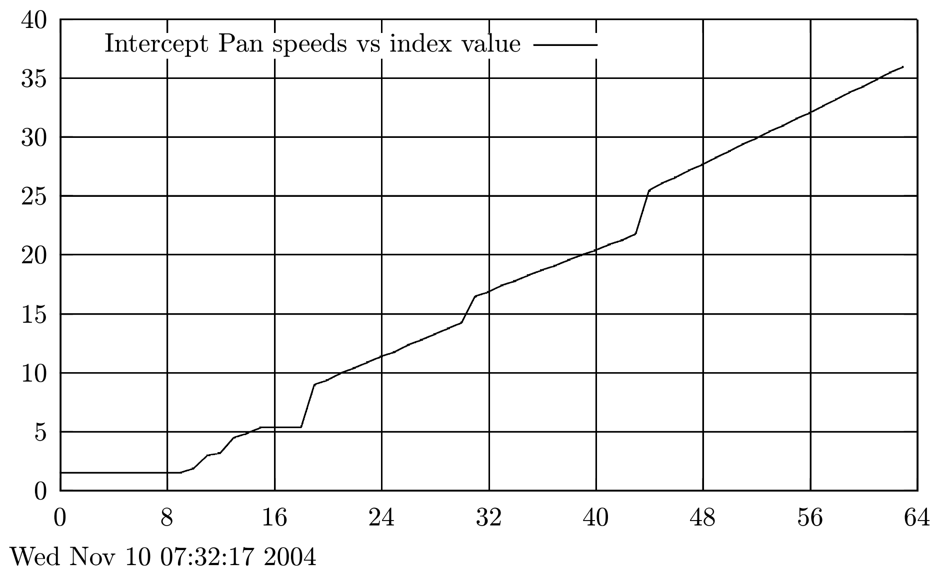


Figure 1. Intercept `DRD08A12u3_R3.06` Pan Speeds

[4] `$Header: d:/UnitSpeeds/RCS/Ispeeds.inc,v 1.12 2004-12-15 07:10:45-08 Hamilton Exp Hamilton $`
[5] `$Header: d:/UnitSpeeds/RCS/IspeedP.inc,v 1.2 2004-11-10 08:27:28-08 Hamilton Exp Hamilton $`

```
/* pan speed translate data for skipping noisy speeds */
/* speed 9 is 1.5 degrees/second, speed 63 is 36 degrees/second
   skip speed ranges (degrees/second) 2.25-3.00, 3.33-4.50,
        6.75-9.00, 14.3-16.5, 21.8-25.5 */
/* last 3 speeds of 4th range changed to 54 as a result of
        testing */
uns code pan_speed_xlat[MAX_NORMAL_SPEED + 1] =
        {
         15,  15,  15,  15,  15,  15,  15,  15,
         15,                                    /* 1st range */
              15,  19,                          /* 2nd range */
                        30,  32,                /* 3rd range */
                                  45,  49,  54,
         54,  54,  54,                          /* 4th range */
                        90,  94, 100, 104, 109,
        114, 118, 124, 128, 133, 138, 143,      /* 5th range */
                                           165,
        169, 174, 178, 183, 187, 191, 196, 200,
        204, 209, 213, 218,                     /* 6th range */
                            255, 261, 266, 272,
        277, 283, 288, 294, 299, 305, 310, 316,
        321, 327, 332, 338, 343, 349, 355, 360  /* 7th range */
        };
```

Table 1. Pan speeds Intercept, version DRD08A12u3_R3.06, from the source code

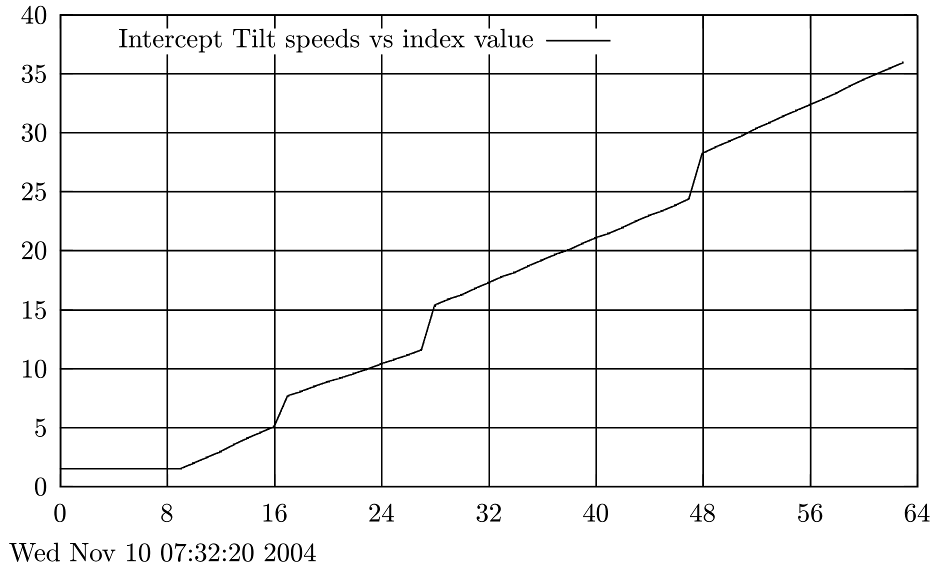


Figure 2. Intercept `DRD08A12u3_R3.06` Tilt Speeds

[6] `$Header: d:/UnitSpeeds/RCS/IspeedT.inc,v 1.2 2004-11-10 08:27:28-08 Hamilton Exp Hamilton $`

```
/* speed 9 is 1.5 degrees/second, speed 63 is 36 degrees/second
        skip speed ranges (degrees/second) 5.14-7.71, 11.6-15.4,
        24.4-28.3 */
uns code tilt_speed_xlat[MAX_NORMAL_SPEED + 1] =
        {
         15,  15,  15,  15,  15,  15,  15,  15,
         15,                                    /* 1st range */
              15,  20,  25,  30,  36,  41,  46,
         51,                                    /* 2nd range */
              77,  81,  85,  89,  92,  96, 100,
        104, 108, 112, 116,                     /* 3rd range */
                            154, 159, 163, 168,
        173, 178, 182, 187, 192, 197, 201, 206,
        211, 215, 220, 225, 230, 234, 239, 244, /* 4th range */
        283, 288, 293, 298, 304, 309, 314, 319,
        324, 329, 334, 340, 345, 350, 355, 360  /* 5th range */
        };
```

Table 2. Tilt speeds Intercept, version `DRD08A12u3_R3.06`, from the source code

1.3 Spectra domes

After manufacturing the Intercept series of domes and gaining knowledge and experience in variable speed drives utilizing stepper motors, Pelco designed and started to produce a fully integrated series of domes called the Spectra line. The Spectra line has gone through and original series and two significant upgrades. The current model is the Spectra III line of domes and consists of several models based on:

1. Camera type, i.e. optical lens power (x16, x22 or x23) and a x10 digital "zoom".
2. Television standard, i.e. NTSC or PAL.
3. The original Spectra I had a x12 optical lens power and a x8 digital "zoom".

With the Spectra series speeds were originally linear with some anti-vibration "jumps". It will be noted in a graphing of the Spectra I pan speeds (Figure 3, page 17) that there are two anomalies near input speed step numbers 13 and 19, while the tilt speeds (Figure 4, page 18) are essentially linear. In both figures also note that there is a "dead band" at the start where all input speeds result in very slow speeds. This is done to compensate for the characteristics of various Pelco keyboards.

The speeds originally used with the Spectra II (Figure 5, page 19 and Figure 7, page 23) that the original tables were very linear.

Shortly the Spectra II speeds were changed to be "predictably non-linear" as is shown in Figure 6, page 21, for pan, and Figure 8, page 24, for tilt[7]. Later in the development cycle the speeds were made more "aggressive" as is shown in Figure 9, page 25, for pan, and Figure 10, page 27, for tilt. It should be noted that the maximum non-turbo pan speed has now been raised from 40^osec to 80^osec. Tilt maximum speed remains the same as before. Some changes were made to the tilt speeds to skip over vibration causing speeds with different speed tables being used for NTSC and PAL television systems[8]

Spectra III uses the same speed tables as the last version of Spectra II.

[7]The change from linear to non-linear speeds were probably were the result of the findings in "The Joystick Report", September 19, 1997. (Section 3, page 43)

[8]There appears to be a coding error in the tilt speed table for PAL near input speed step number 59. This anomaly is shown in Figure 12, page 29. It is unknown how this got into the code.

1.3.1 Spectra I, version `DD5x- PRGSPCTFW106`

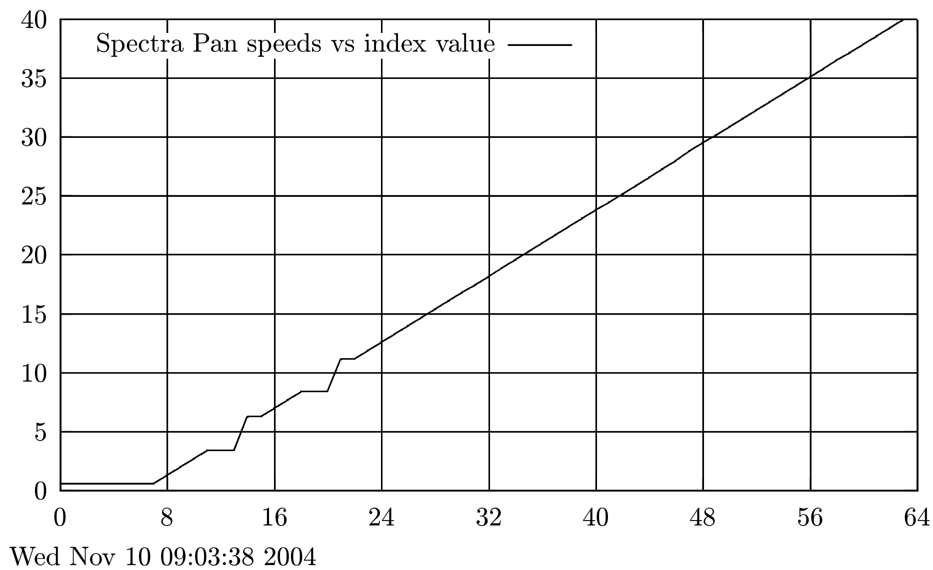


Figure 3. Spectra I `DD5x- PRGSPCTFW106` Pan Speeds

```
static code WORD pan_speed_xlat[] = {
         6,   6,   6,   6,   6,   6,   6,   6,
        13,  20,  27,  34,  34,  34,  63,  63, /* skip speed 12(41), 13(48),
                                                 and 14(56) */
        70,  77,  84,  84,  84, 112, 112, 119, /* skip speed 19(91), 20(98),
                                                 and 21(105) */
       126, 133, 140, 147, 154, 161, 168, 175,
       182, 189, 196, 203, 210, 217, 224, 231,
       238, 245, 252, 259, 266, 273, 280, 288,
       295, 302, 309, 316, 323, 330, 337, 344,
       351, 358, 365, 372, 379, 386, 393, 400
};
```

Table 3. Pan speeds Spectra I, version `DD5x- PRGSPCTFW106`, from the source code

[9] `$Header: d:/UnitSpeeds/RCS/Sspeeds.inc,v 1.13 2004-11-10 10:16:24-08 Hamilton Exp Hamilton $`
[10] `$Header: d:/UnitSpeeds/RCS/SspeedP1.inc,v 1.1 2004-11-10 09:43:59-08 Hamilton Exp Hamilton $`

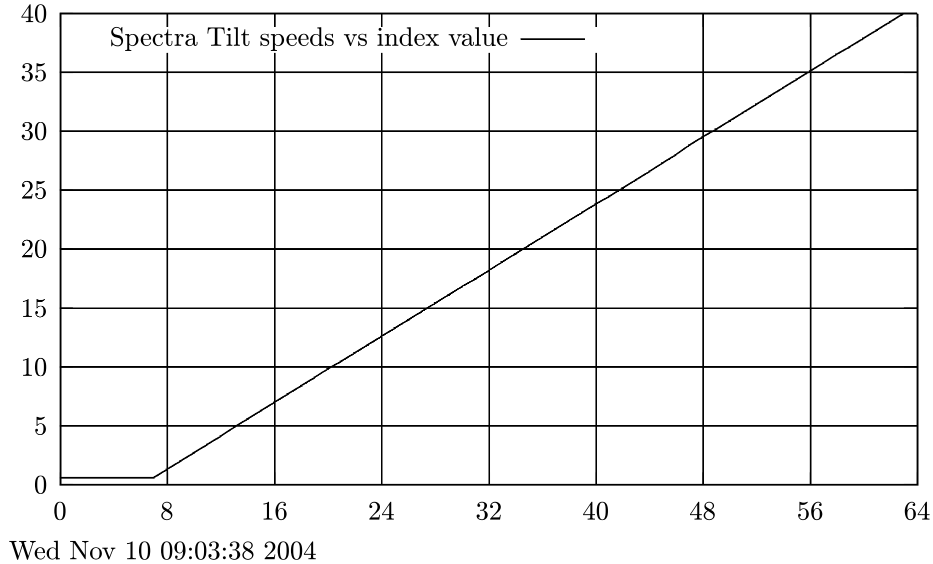


Figure 4. Spectra I DD5x- PRGSPCTFW106 Tilt Speeds

```
/* minimum input speed 7 */
static code WORD tilt_speed_xlat[] = {
          6,   6,   6,   6,   6,   6,   6,   6,
         13,  20,  27,  34,  41,  49,  56,  63,
         70,  77,  84,  91,  98, 105, 112, 119,
        126, 133, 140, 147, 154, 161, 168, 175,
        182, 189, 196, 203, 210, 217, 224, 231,
        238, 245, 252, 259, 266, 273, 280, 288,
        295, 302, 309, 316, 323, 330, 337, 344,
        351, 358, 365, 372, 379, 386, 393, 400
  };
```

Table 4. Tilt speeds Spectra I, version DD5x- PRGSPCTFW106, from the source code

[11] $Header: d:/UnitSpeeds/RCS/SspeedT1.inc,v 1.1 2004-11-10 09:44:00-08 Hamilton Exp Hamilton $

1.3.2 Spectra II, version `PG53-0001-0206`

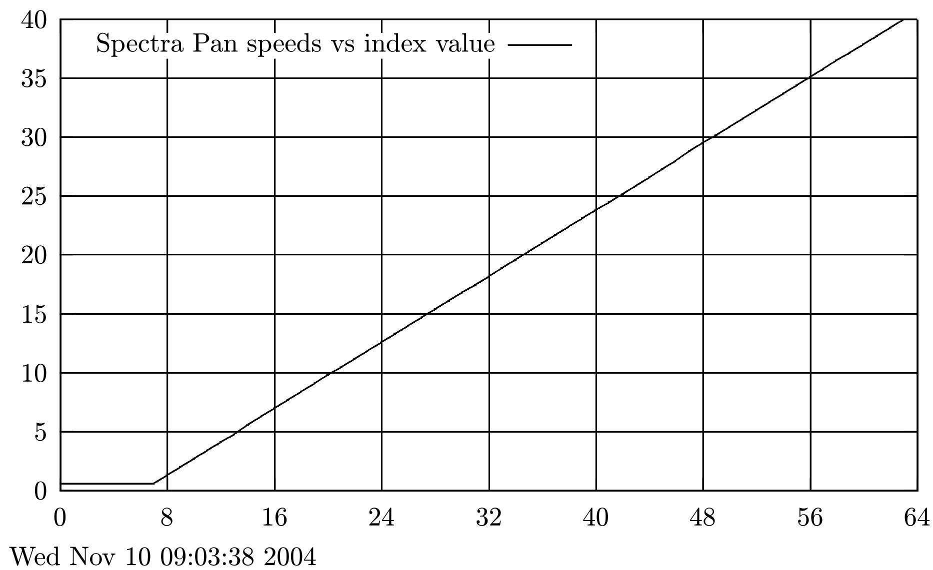


Figure 5. Spectra II `PG53-0001-0206` "0206" Pan Speeds"OLD_SPEED_TABLE"

```
#ifdef OLD_SPEED_TABLE /* 2.00 */
static code WORD pan_speed_xlat[] = {
          6,   6,   6,   6,   6,   6,   6,   6,
         13,  20,  27,  34,  41,  48,  56,  63, /* 2.00 */
         70,  77,  84,  91,  98, 105, 112, 119, /* 2.00 */
        126, 133, 140, 147, 154, 161, 168, 175,
        182, 189, 196, 203, 210, 217, 224, 231,
        238, 245, 252, 259, 266, 273, 280, 288,
        295, 302, 309, 316, 323, 330, 337, 344,
        351, 358, 365, 372, 379, 386, 393, 400
  };
#else /* 2.00 */
#endif /* 2.00 */
/* End of 2.00 */
```

[12]`$Header: d:/UnitSpeeds/RCS/Sspeeds.inc,v 1.13 2004-11-10 10:16:24-08 Hamilton Exp Hamilton $`
[13]`$Header: d:/UnitSpeeds/RCS/SspeedP2.inc,v 1.1 2004-11-10 09:43:59-08 Hamilton Exp Hamilton $`

Table 5. Pan speeds Spectra II, version `PG53-0001-0206`, from the source code, first part

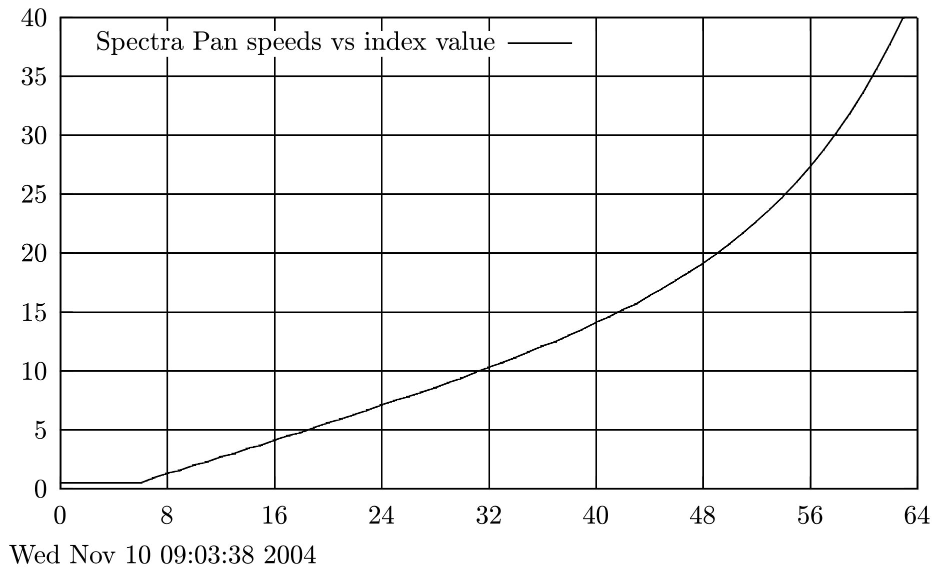


Figure 6. Spectra II `PG53-0001-0206` "0206" Pan Speeds "non-OLD_SPEED_TABLE"

```
#ifdef OLD_SPEED_TABLE /* 2.00 */
/* Start of 2.00 */
static code WORD pan_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
  };
#endif /* 2.00 */

static code WORD pan_jitter_speeds[] =
  {
  34, 48, 63,
  84, 98, 112
  };
#define NPAN_JITTER_SPEEDS (sizeof(pan_jitter_speeds) / sizeof (WORD))
```

[14] `$Header: d:/UnitSpeeds/RCS/SspeedP3.inc,v 1.1 2004-11-10 09:44:00-08 Hamilton Exp Hamilton $`

```
/* End of 2.00 */
```

Table 6. Pan speeds Spectra II, version `PG53-0001-0206`, from the source code, second part

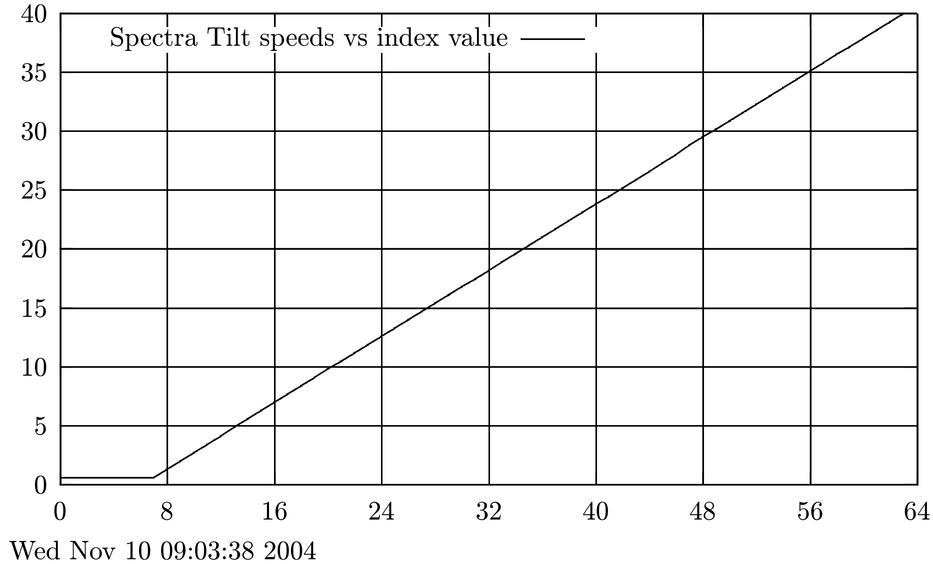


Figure 7. Spectra II PG53-0001-0206 "0206" Tilt Speeds "OLD_SPEED_TABLE"

```
#ifdef OLD_SPEED_TABLE
/* minimum input speed 7 */
static code WORD tilt_speed_xlat[] = {
          6,   6,   6,   6,   6,   6,   6,   6,
         13,  20,  27,  34,  41,  49,  56,  63,
         70,  77,  84,  91,  98, 105, 112, 119,
        126, 133, 140, 147, 154, 161, 168, 175,
        182, 189, 196, 203, 210, 217, 224, 231,
        238, 245, 252, 259, 266, 273, 280, 288,
        295, 302, 309, 316, 323, 330, 337, 344,
        351, 358, 365, 372, 379, 386, 393, 400
  };
#else
#endif /* 2.00 */
```

Table 7. Tilt speeds Spectra II, version PG53-0001-0206, from the source code, first part

[15] $Header: d:/UnitSpeeds/RCS/SspeedT2.inc,v 1.1 2004-11-10 09:44:00-08 Hamilton Exp Hamilton $

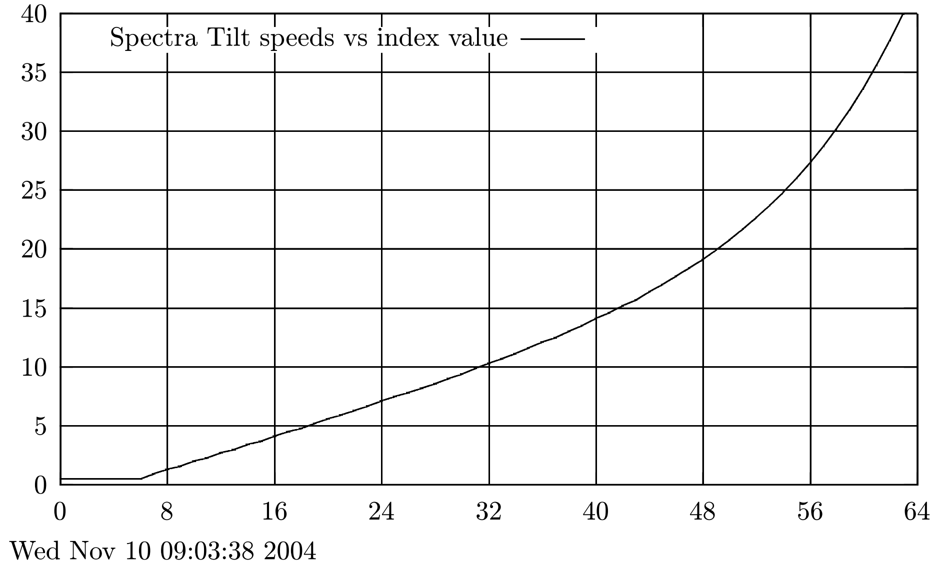


Figure 8. Spectra II PG53-0001-0206 “0206” Tilt Speeds “non-OLD_SPEED_TABLE”

```
#ifdef OLD_SPEED_TABLE
#else
static code WORD tilt_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
  };
#endif /* 2.00 */
```

Table 8. Tilt speeds Spectra II, version PG53-0001-0206, from the source code, second part

[16] $Header: d:/UnitSpeeds/RCS/SspeedT3.inc,v 1.1 2004-11-10 09:44:01-08 Hamilton Exp Hamilton $

1.3.3 Spectra II, version `PG53-0060-0308`

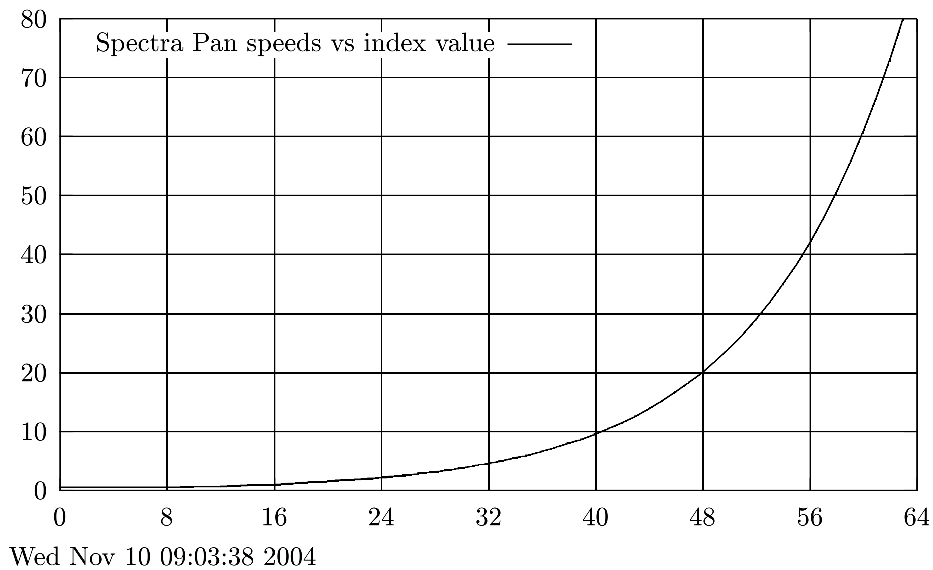


Figure 9: Spectra II `PG53-0060-0308` "0308" Pan Speeds. **Note:** that the vertical scaling has been changed.

```
static code WORD pan_speed_xlat[] = {
           5,   5,   5,   5,   5,   5,   5,   5,
           5,   5,   6,   7,   7,   8,   9,  10,
          10,  11,  13,  14,  15,  17,  18,  20,
          22,  24,  26,  29,  32,  35,  38,  42,
          46,  50,  55,  60,  66,  73,  80,  87,
          96, 105, 115, 126, 139, 152, 167, 183,
         200, 220, 241, 264, 290, 318, 349, 382,
         419, 460, 504, 553, 607, 665, 729, 800
  };

#endif /* 2.10 */
/* Line removed 2.10 */

#ifdef SKIP_VIBRATION_SPEEDS /* 2.10 */
static code WORD pan_jitter_speeds[] =
```

[17] `$Header: d:/UnitSpeeds/RCS/Sspeeds.inc,v 1.13 2004-11-10 10:16:24-08 Hamilton Exp Hamilton $`
[18] `$Header: d:/UnitSpeeds/RCS/SspeedP4.inc,v 1.1 2004-11-10 09:44:00-08 Hamilton Exp Hamilton $`

```
  {
        34, 48, 63,
        84, 98, 112
  };
#define NPAN_JITTER_SPEEDS (sizeof(pan_jitter_speeds) / sizeof (WORD))
#endif /* 2.10 */

/* End of 2.00 */
```

Table 9. Pan speeds Spectra II, version `PG53-0060-0308`, from the source code

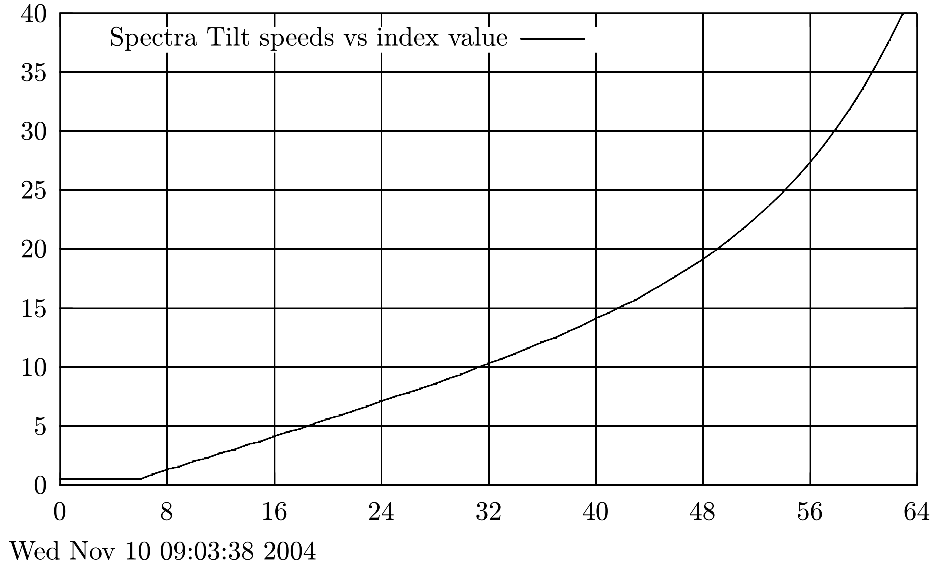


Figure 10. Spectra II `PG53-0060-0308` "0308" Tilt Speeds

```
/* Lines removed 2.10 */
/* Start of 2.00 */
static code WORD tilt_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
  };

/* End of 2.00 */
/* Line removed 2.10 */
```

Table 10. Tilt speeds Spectra II, version `PG53-0060-0308`, from the source code

[19] `$Header:  d:/UnitSpeeds/RCS/SspeedT4.inc,v 1.1 2004-11-10 09:44:01-08 Hamilton Exp Hamilton $`

1.3.4 Spectra II, version `PG53-0060-0331`

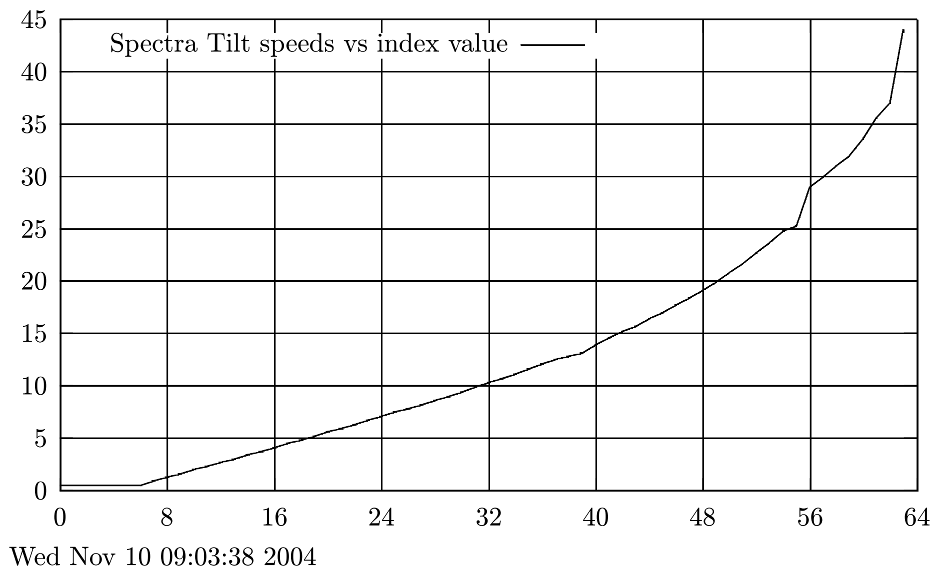


Figure 11. Spectra II `PG53-0060-0331` "0331" NTSC Tilt Speeds

```
static code WORD tilt_ntsc_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 128, 131,
        139, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 253,
        290, 300, 310, 320, 336, 356, 370, 440
};
```

Table 11. Tilt speeds Spectra II, version `PG53-0060-0331`, from the source code, first part

[20] `$Header: d:/UnitSpeeds/RCS/Sspeeds.inc,v 1.13 2004-11-10 10:16:24-08 Hamilton Exp Hamilton $`
[21] `$Header: d:/UnitSpeeds/RCS/SspeedT5.inc,v 1.1 2004-11-10 09:44:01-08 Hamilton Exp Hamilton $`

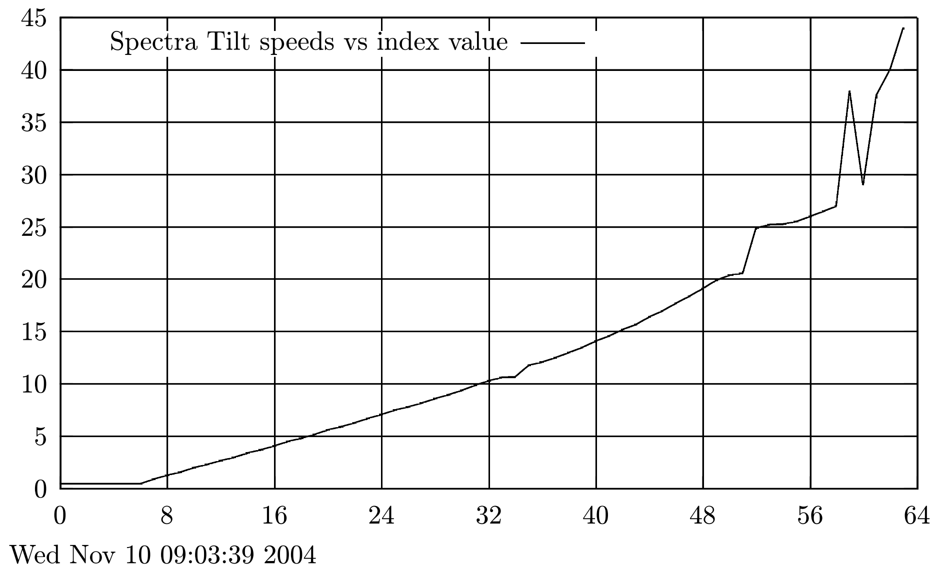


Figure 12. Spectra II `PG53-0060-0331` "0331" PAL Tilt Speeds

```
static code WORD tilt_pal_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 106, 107, 118, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 204, 206, 249, 252, 253, 255,
        260, 265, 270, 380, 290, 376, 400, 440
};

/*
   for version 3.29:
   ntsc:
   gap inserted between 131 and 139, try to skip 135
   gap inserted between 253 and 290, try to skip 270
   gap inserted between 377 and 440, try to skip 405
   pal:
   gap inserted between 107 and 118, try to skip 112
   gap inserted between 206 and 249, try to skip 224
```

[22] `$Header: d:/UnitSpeeds/RCS/SspeedT6.inc,v 1.1 2004-11-10 09:44:01-08 Hamilton Exp Hamilton $`

```
   gap inserted between 290 and 376, try to skip 336
*/
```

Table 12. Tilt speeds Spectra II, version PG53-0060-0331, from the source code, second part

1.4 Esprit speeds

The Esprit series of pan and tilt units was based on Spectra II source code and shows reasonable non-linear speed curves for both pan (Figure 13, page 31 and the very similar Figure 14, page 32). The tilt speeds were partially from the Esprit tilt speed table (Figure 15, page 33) and an Esprit specific tilt speed table (Figure 16, page 34).

Eventually the Esprit updated their speed tables to be reasonably non-linear with Figure 17, page 35, for pan, and Figure 18, page 36 for tilt.

1.4.1 Esprit, version `PG53-0026-0100`

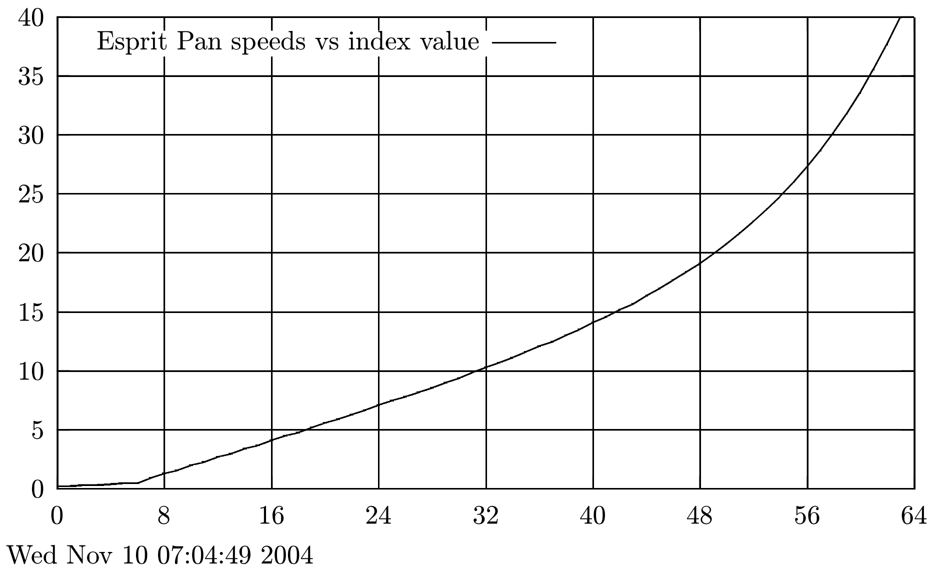


Figure 13. Esprit `PG53-0026-0100` Pan Speeds "NOTEST"

```
static code WORD pan_speed_xlat[] = {
#ifdef NOTEST
          2,   2,   3,   3,   4,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
#else
#endif
        };
```

Table 13. Pan speeds Esprit, version `PG53-0026-0100`, from the source code, first part

[23] `$Header: d:/UnitSpeeds/RCS/Espeeds.inc,v 1.10 2004-11-10 08:27:26-08 Hamilton Exp Hamilton $`
[24] `$Header: d:/UnitSpeeds/RCS/EspeedP1.inc,v 1.2 2004-11-10 08:27:24-08 Hamilton Exp Hamilton $`

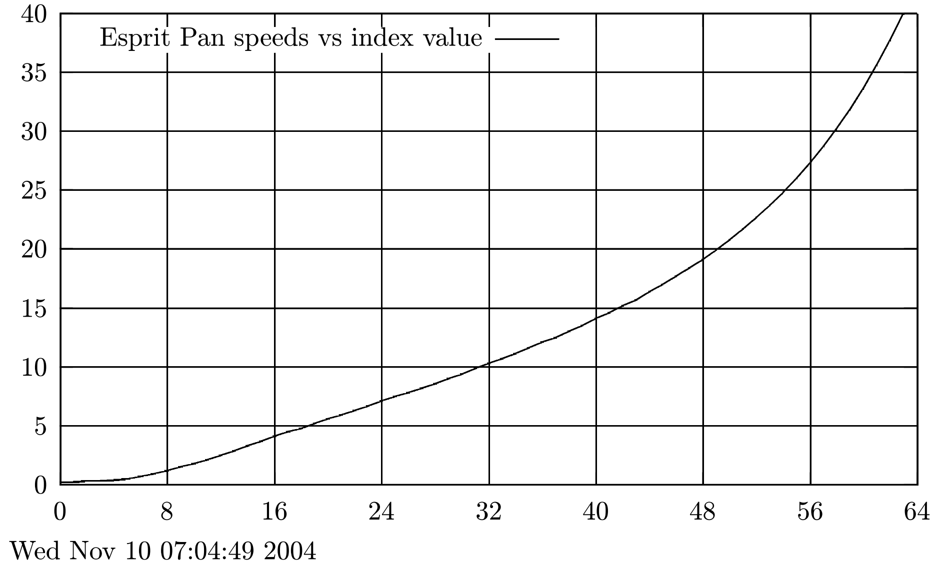


Figure 14. Esprit PG53-0026-0100 Pan Speeds “not-NOTEST”

```
static code WORD pan_speed_xlat[] = {
#ifdef NOTEST
#else
          2,   2,   3,   3,   4,   5,   7,   9,
         12,  15,  18,  21,  25,  29,  33,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
#endif
        };
```

Table 14. Pan speeds Esprit, version PG53-0026-0100, from the source code, second part

[25] $Header: d:/UnitSpeeds/RCS/EspeedP2.inc,v 1.2 2004-11-10 08:27:25-08 Hamilton Exp Hamilton $

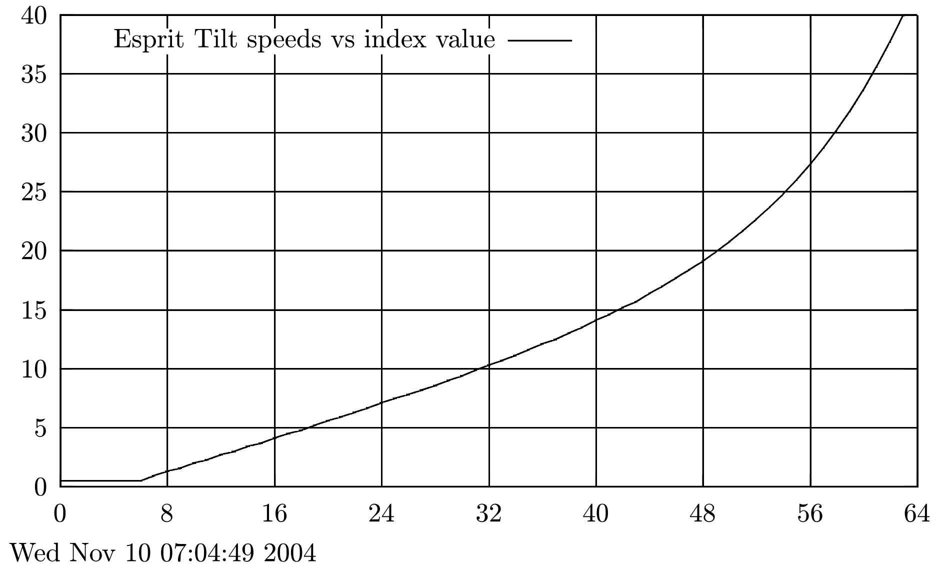


Figure 15. Esprit PG53-0026-0100 Tilt Speeds “OLD_SPEED_TABLE”

```
#ifdef OLD_SPEED_TABLE /* v0.09 */
static code WORD tilt_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   9,
         13,  16,  20,  23,  27,  30,  34,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
        };
#else /* v0.09 */
#endif /* v0.09 */
```

Table 15. Tilt speeds Esprit, version PG53-0026-0100, from the source code, first part

[26] $Header: d:/UnitSpeeds/RCS/EspeedT1.inc,v 1.2 2004-11-10 08:27:25-08 Hamilton Exp Hamilton $

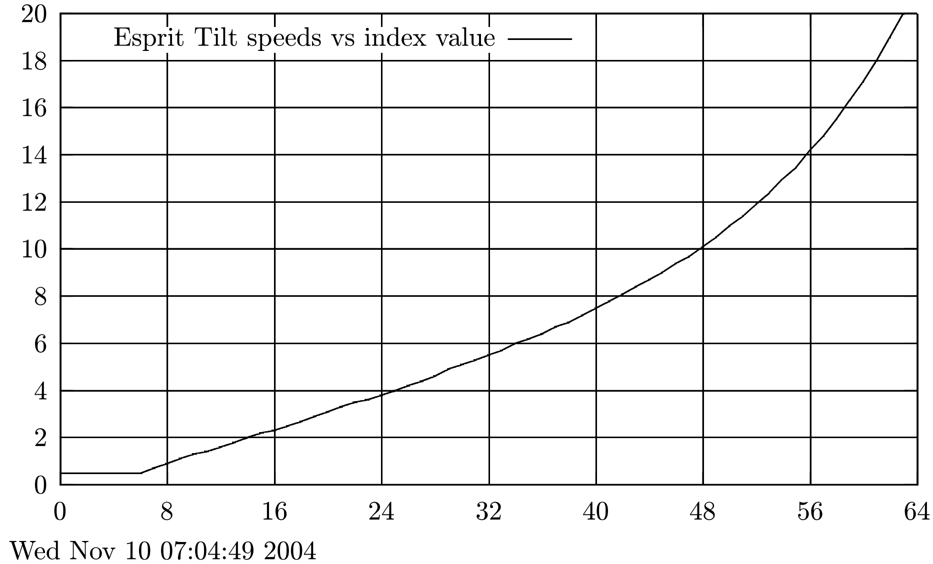


Figure 16. Esprit PG53-0026-0100 Tilt Speeds "not-OLD_SPEED_TABLE"

```
#ifdef OLD_SPEED_TABLE /* v0.09 */
#else /* v0.09 */
/* each step */
static code WORD tilt_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   7,
          9,  11,  13,  14,  16,  18,  20,  22,
         23,  25,  27,  29,  31,  33,  35,  36,
         38,  40,  42,  44,  46,  49,  51,  53,
         55,  57,  60,  62,  64,  67,  69,  72,
         75,  78,  81,  84,  87,  90,  94,  97,
        101, 105, 110, 114, 119, 124, 130, 135,
        142, 148, 155, 163, 171, 180, 190, 200
      }
#endif /* v0.09 */
```

Table 16. Tilt speeds Esprit, version PG53-0026-0100, from the source code, second part

[27] $Header: d:/UnitSpeeds/RCS/EspeedT2.inc,v 1.2 2004-11-10 08:27:26-08 Hamilton Exp Hamilton $

1.4.2 Esprit, version PG53-0096-0210

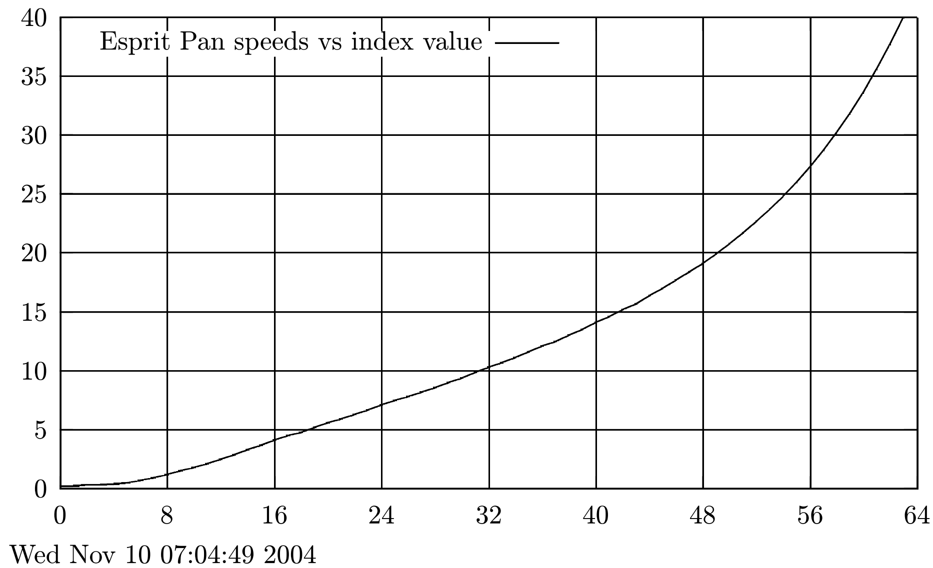


Figure 17. Esprit PG53-0096-0210 Pan Speeds

```
static code WORD pan_speed_xlat[] = {
          2,   2,   3,   3,   4,   5,   7,   9,
         12,  15,  18,  21,  25,  29,  33,  37,
         41,  45,  48,  52,  56,  59,  63,  67,
         71,  75,  78,  82,  86,  90,  94,  99,
        103, 107, 111, 116, 121, 125, 130, 135,
        141, 146, 152, 157, 164, 170, 177, 184,
        191, 199, 208, 217, 227, 237, 248, 260,
        273, 287, 302, 318, 336, 356, 377, 400
};
```

Table 17. Pan speeds Esprit, version PG53-0096-0210, from the source code

[28]$Header: d:/UnitSpeeds/RCS/Espeeds.inc,v 1.10 2004-11-10 08:27:26-08 Hamilton Exp Hamilton $
[29]$Header: d:/UnitSpeeds/RCS/EspeedP3.inc,v 1.2 2004-11-10 08:27:25-08 Hamilton Exp Hamilton $

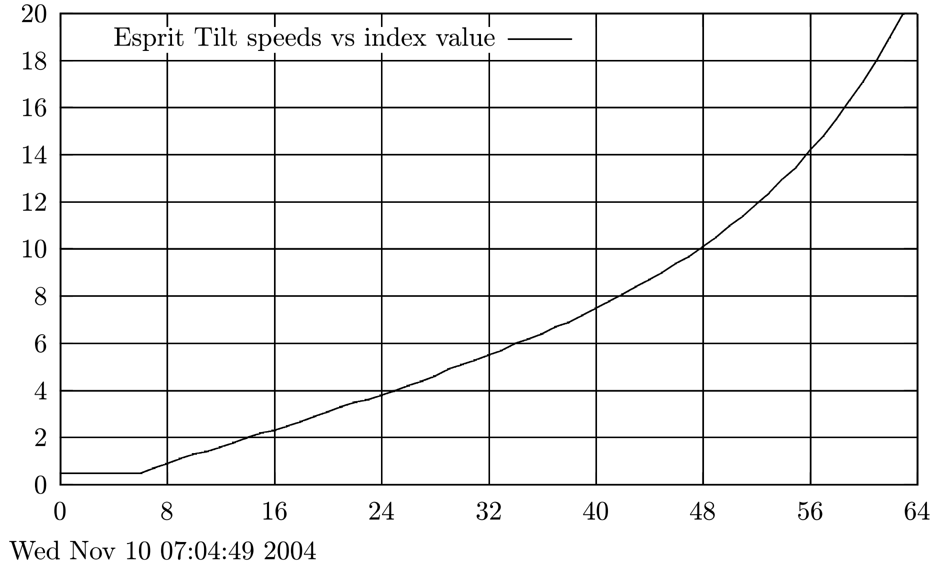


Figure 18. Esprit `PG53-0096-0210` Tilt Speeds

```
static code WORD tilt_speed_xlat[] = {
          5,   5,   5,   5,   5,   5,   5,   7,
          9,  11,  13,  14,  16,  18,  20,  22,
         23,  25,  27,  29,  31,  33,  35,  36,
         38,  40,  42,  44,  46,  49,  51,  53,
         55,  57,  60,  62,  64,  67,  69,  72,
         75,  78,  81,  84,  87,  90,  94,  97,
        101, 105, 110, 114, 119, 124, 130, 135,
        142, 148, 155, 163, 171, 180, 190, 200
};
```

Table 18. Tilt speeds Esprit, version `PG53-0096-0210`, from the source code

[30] `$Header:  d:/UnitSpeeds/RCS/EspeedT3.inc,v 1.2 2004-11-10 08:27:26-08 Hamilton Exp Hamilton $`

1.4.3 Esprit, version PG53-0026-0100

The ExCite system has identical tilt speed tables for tilt in NTSC and PAL modes of operation.

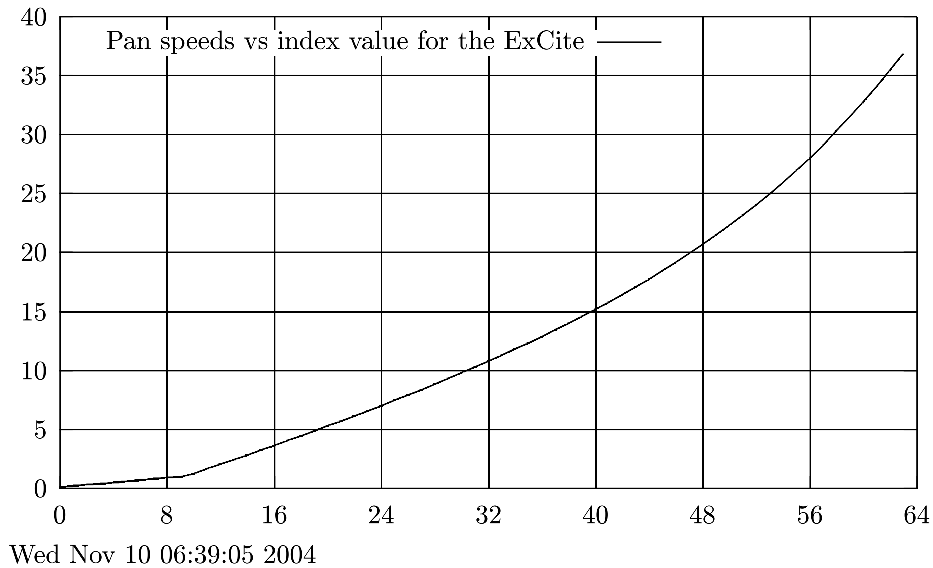


Figure 19. ExCite pan speeds, first released version

```
#ifdef MAX_SPEED_80

LOCAL
    #ifndef BB_SPEED_TABLE
const
    #endif
unsigned short pan_speed_xlat[] = {
  10,   20,   30,   40,   50,   60,   70,   80,   90,  100,  127,  166,
 205,  245,  284,  324,  365,  405,  446,  488,  530,  572,  615,  658,
 702,  747,  792,  838,  884,  931,  980, 1029, 1078, 1129, 1181, 1234,
1289, 1344, 1401, 1459, 1519, 1581, 1644, 1710, 1777, 1846, 1918, 1992,
2069, 2149, 2231, 2317, 2406, 2499, 2595, 2696, 2801, 2911, 3026, 3146,
3272, 3403, 3542, 3687
};
#endif
```

[31] $Header: d:/UnitSpeeds/RCS/ExCite.inc,v 1.2 2004-11-10 10:33:41-08 Hamilton Exp Hamilton $
[32] $Header: d:/UnitSpeeds/RCS/ExCiteP.inc,v 1.2 2004-11-10 10:33:41-08 Hamilton Exp Hamilton $

Table 19: Pan and Tilt speeds for the initial version of the ExCite, from the source code, first part

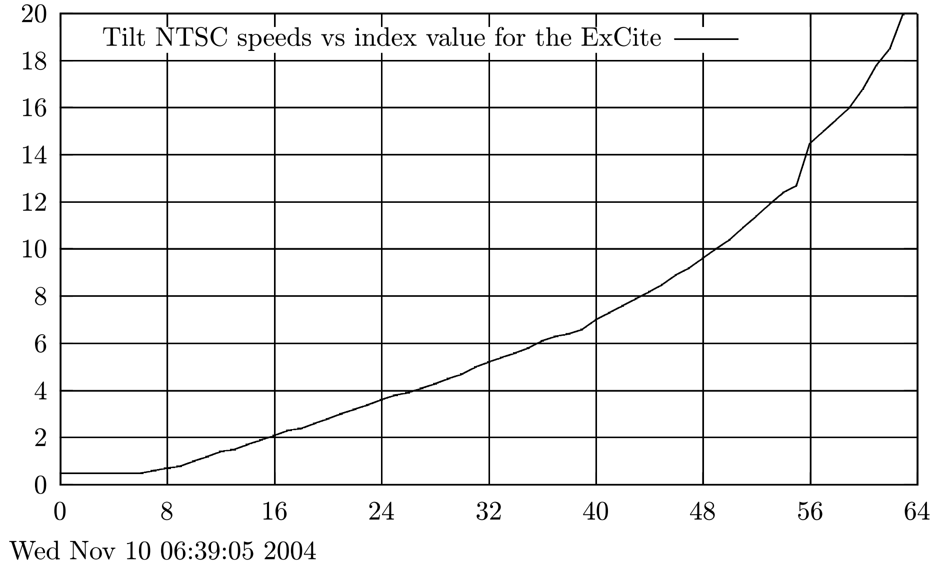


Figure 20. ExCite NTSC tilt speeds, first release

```
LOCAL
#ifndef BB_SPEED_TABLE
const
#endif
unsigned short tilt_ntsc_xlat[] = {
   50,   50,   50,   50,   50,   50,   50,   60,   70,   80, 100, 120,
  140,  150,  170,  190,  210,  230,  240,  260,  280,  300, 320, 340,
  360,  380,  390,  410,  430,  450,  470,  500,  520,  540, 560, 580,
  610,  630,  640,  660,  700,  730,  760,  790,  820,  850, 890, 920,
  960, 1000, 1040, 1090, 1140, 1190, 1240, 1270, 1450, 1500, 1550, 1600,
 1680, 1780, 1850, 2000
};
```

Table 20. Pan and Tilt speeds for the initial version of the ExCite, from the source code

[33] `$Header: d:/UnitSpeeds/RCS/ExCiteTN.inc,v 1.2 2004-11-10 10:33:42-08 Hamilton Exp Hamilton $`

2 The PMD/UMDM/PUD motion control chips

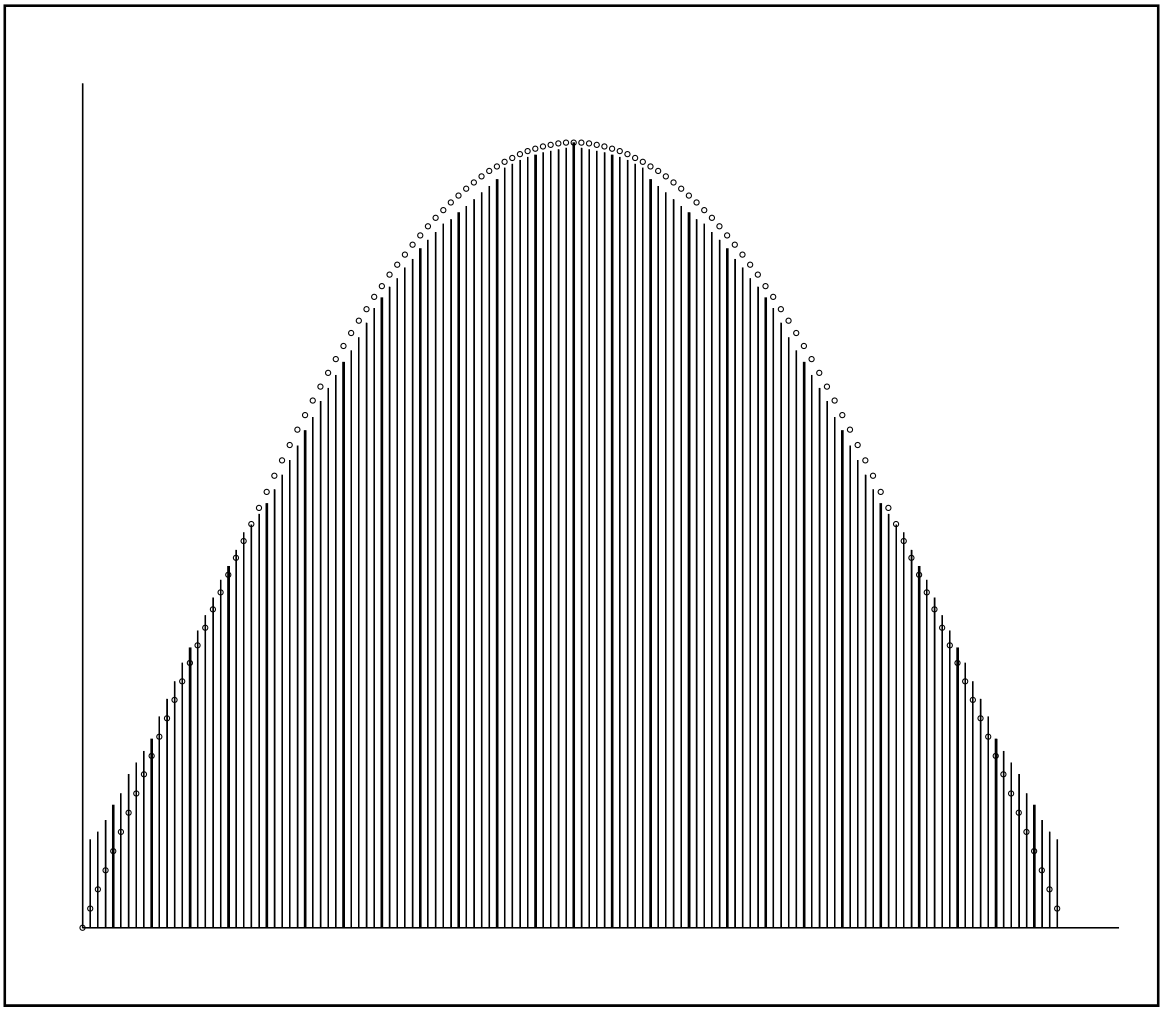
With the Intercept series, motion control was either fixed speed motion control or by "half stepping" stepper motors and utilizing a large pan ratio to get good slow speed. The control signals for the half stepping of the stepper motors were generated by the Intercept software with an assist from some special firmware in the programmable Xilinx chip.

When the design of the Spectra was started it was realized that better smooth motion control could be obtained by "microstepping" the stepper motors. Eventually a motion control chip set (two chips) made by Performance Motor Devices (**PMD**) was chosen to do the microstepping. The PMD chip provides 64 microsteps per "full step", velocity and trajectory control for the system. The interface to it consists of sending it commands and parameters. It then generates the correct Pulse Width Modulated (**PWM**) signal to drive two stepper motors. (One for pan and one for tilt.) (This chip set is still used in all Spectra and some Esprit units today.)

In an effort to control our own products, Pelco in 1997 started a project to make our "own" motion control chips. This was originally known as the Universal Motor Driver Module (**UMDM**) project and has resulted in one patent (Appendix A.1, page A-1). Later on the name was changed to the Pelco Universal Driver (**PUD**) project which is currently being used in some Esprits units. The PUD resulted in one patent (Appendix A.2, page A-1).

A feature of the PUD chips is that they may have some of their internal tables downloaded. These are the $sin\Theta$ table (Figure 21, page 41) used to generate the motor control data and the IVcorrection table to provide better control of the stepper motor at high speeds. The downloaded $sin\Theta$ table from one of the Espirts is shown in Figure 2.1, page 41.

2.1 sin table used with the PUD from the Esprit `PG53-0096-0210`



$RCSfile: PudSin.inc,v $

Figure 21. PUD sin wave *vs.* a "real" one

[34] `$Header: d:/UnitSpeeds/RCS/PudSin.inc,v 1.6 2004-11-09 14:38:01-08 Hamilton Exp Hamilton $`

```
static code unsigned int pansine[SINE_TABLE_SIZE]
    = {
      0,   115,  125,  140,  160,  175,  200,  215,
    230,   246,  275,  298,  321,  345,  365,  387,
    407,   430,  453,  471,  492,  515,  525,  539,
    553,   571,  590,  609,  628,  648,  665,  686,
    703,   720,  737,  752,  769,  788,  807,  821,
    835,   846,  860,  871,  885,  896,  906,  917,
    923,   932,  940,  949,  958,  966,  975,  990,
    995,  1000, 1004, 1007, 1010, 1012, 1014, 1016,
   1023,  1016, 1014, 1012, 1010, 1007, 1004, 1000,
    995,   990,  975,  966,  958,  949,  940,  932,
    923,   917,  906,  896,  885,  871,  860,  846,
    835,   821,  807,  788,  769,  752,  737,  720,
    703,   686,  665,  648,  628,  609,  590,  571,
    553,   539,  525,  515,  492,  471,  453,  430,
    407,   387,  365,  345,  321,  298,  275,  246,
    230,   215,  200,  175,  160,  140,  125,  115,
};

static code unsigned int tiltsine[SINE_TABLE_SIZE]
    = {
      0,   115,  125,  140,  160,  175,  200,  215,
    230,   246,  275,  298,  321,  345,  365,  387,
    407,   430,  453,  471,  492,  515,  525,  539,
    553,   571,  590,  609,  628,  648,  665,  686,
    703,   720,  737,  752,  769,  788,  807,  821,
    835,   846,  860,  871,  885,  896,  906,  917,
    923,   932,  940,  949,  958,  966,  975,  990,
    995,  1000, 1004, 1007, 1010, 1012, 1014, 1016,
   1023,  1016, 1014, 1012, 1010, 1007, 1004, 1000,
    995,   990,  975,  966,  958,  949,  940,  932,
    923,   917,  906,  896,  885,  871,  860,  846,
    835,   821,  807,  788,  769,  752,  737,  720,
    703,   686,  665,  648,  628,  609,  590,  571,
    553,   539,  525,  515,  492,  471,  453,  430,
    407,   387,  365,  345,  321,  298,  275,  246,
    230,   215,  200,  175,  160,  140,  125,  115,
};

"Real sin() table"
          0,    25,   50,   75,  100,  125,  150,  175,
        200,   224,  249,  273,  297,  321,  345,  368,
        391,   415,  437,  460,  482,  504,  526,  547,
        568,   589,  609,  629,  649,  668,  687,  705,
        723,   741,  758,  775,  791,  806,  822,  836,
        851,   864,  877,  890,  902,  914,  925,  935,
        945,   954,  963,  971,  979,  986,  992,  998,
       1003,  1008, 1012, 1015, 1018, 1020, 1022, 1023,
       1023,  1023, 1022, 1020, 1018, 1015, 1012, 1008,
       1003,   998,  992,  986,  979,  971,  963,  954,
        945,   935,  925,  914,  902,  890,  877,  864,
        851,   836,  822,  806,  791,  775,  758,  741,
        723,   705,  687,  668,  649,  629,  609,  589,
        568,   547,  526,  504,  482,  460,  437,  415,
        391,   368,  345,  321,  297,  273,  249,  224,
        200,   175,  150,  125,  100,   75,   50,   25,
```

3 About the Joystick Report

The original source for the Joystick Report has been misplaced[35]. There are several "Xerox" copies of one of the originals. The overall findings of the Joystick Report are that a non-linear speed table resident on the dome, pan/tilt, should be used for the best control of a dome or pan/tilt device. For this to work correctly the commands received by the dome, **must** be generated in a linear manner. I.e. as the joy stick is moved further and further from the center the values generated must monotonically increase in a linear manner. Since humans perceive most things in a non-linear manner, the non-linear portion of the control loop must be inside the dome and must be matched to the dome's internal/physical characteristics. A suggested table of non-linear dome speeds is shown in Table 21, page 45 of the Joystick Report, it is plotted in Figure 22, page 44. Motion control of the Spectra series of domes improved significantly after the recommendations of the Joystick Report were adopted. (The Esprit series of pan/tilt units always have had the improved speed tables suggested by the Joystick Report.)

[35] The last known original was "misplaced" by Eric Hamilton sometime in this century or the last.

3.1 "Ideal" speeds from sheet 6 of the Joystick Report

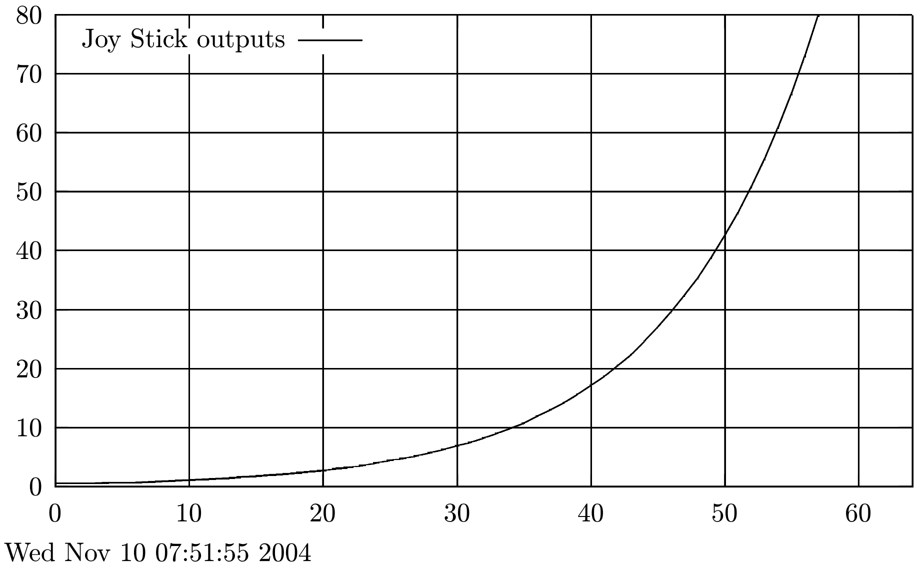


Figure 22. Ideal Joystick speeds

[36]$Header: d:/UnitSpeeds/RCS/JsSpeeds.inc,v 1.4 2004-11-10 08:27:28-08 Hamilton Exp Hamilton $
[37]$Header: d:/UnitSpeeds/RCS/Jspeeds.inc,v 1.2 2004-11-10 08:27:29-08 Hamilton Exp Hamilton $

1	005	30	016	59	063	88	224
2	005	31	017	60	063	89	246
3	005	32	017	61	069	90	246
4	005	33	019	62	069	91	269
5	005	34	019	63	075	92	269
6	005	35	021	64	075	93	295
7	005	36	021	65	083	94	295
8	005	37	023	66	083	95	323
9	006	38	023	67	090	96	323
10	006	39	025	68	090	97	353
11	007	40	025	69	099	98	353
12	007	41	027	70	099	99	387
13	007	42	027	71	108	100	387
14	007	43	030	72	108	101	424
15	008	44	030	73	119	102	424
16	008	45	033	74	119	103	464
17	009	46	033	75	130	104	464
18	009	47	036	76	130	105	508
19	010	48	036	77	142	106	508
20	010	49	040	78	142	107	556
21	011	50	040	79	156	108	556
22	011	51	044	80	156	109	609
23	012	52	044	81	171	110	609
24	012	53	048	82	171	111	667
25	013	54	048	83	187	112	667
26	013	55	052	84	187	113	730
27	014	56	052	85	205	114	730
28	014	57	057	86	205	115	800
29	016	58	057	87	224	116	800

Table 21. Speed values from sheet 6 of "The Joystick Report", September 19, 1997

4 Firmware Functional Specification for the DRD08/14 series of domes

David Micon, March 27, 1995
David Micon, July 20, 1995
Modified by:
Eric Hamilton, January 2004

4.1 Introduction

This documentation applies to at least one model of the old Intercept series of domes. The Spectra I and following integrated position systems used this, and its upgrades, to as a basis for their internal logic. This write up is "out of date", but is the most recent write up that covers all the internal routines of this code.

This documentation specification applies to several different versions of the software. When this software was writen Pelco had different versions of the software for each of the communications protocols used to control the domes. The protocols were/are:

1. **Coaxitron**. Used to send commands on the same coaxial cable used to send the video from the dome (but in the opposite direction).
2. **Pacom Intercept**, now known as **P Protocol**. Used to communicate on an RS-422/RS-485 connection between the head end and the dome.
3. **American Dynamics**, now known as **D Protocol**. Used to communicate on an RS-422/RS-485 connection between the head end and the dome.

All of these refer to the logical method used to communicate between the control and the Intercept.

The older Intercept domes (there were 8 and 14 inch versions) behavee identically except that the 8-inch version moves twice as fast as the 14-inch version.

In some places in this document there will be refrences to different versions of the Intercept dome.

- The **A22** versions always have video capability (the ability to put messages on the monitor that is connected to the Intercept).
- The **P22** and **D22** versions may or may not have video capability depending on whether or not video hardware is installed.

The following descriptions refer to all models of the Intercept except where noted.

[38] `$Header: d:/UnitSpeeds/RCS/IntLogic.inc,v 1.5 2004-11-09 14:37:59-08 Hamilton Exp Hamilton $`

4.2 Video

Models of the Intercept that have video capability can put two lines of 20 characters each on the monitor. The first line is used for pattern messages and zone labels. The second line is used for preset labels. Pattern messages and preset labels are normally cleared when the Intercept moves (including the lens). Zone labels are updated as the Intercept pans (if zone scan is turned on).

The control can also write characters to the video. This is used to write labels when setting preset and zone labels. When this command is recieved, the monitor is locked so that motion commands will not clear the monitor. This allows a label to be written on the monitor, and then movement to a preset or zone position before setting the preset or zone.

4.3 Firmware Organization

The firmware is organized into a main loop and several tasks. The main loop calls the tasks repeatedly. If a task has nothing to do or is waiting for something to happen, it returns to the main loop quickly. If a task does have something to do, it does it and then returns.

The tasks are divided into six groups: configuration/command, speed ramping, pattern processing, alarm checking, screen refreshing, and zone processing. The configuration/command group has two tasks, configuration processing and command processing. Only one of the tasks will be called during each pass through the main loop, since the Intercept can be either configuring or processing commands but not both at the same time. All the other groups have one task each.

4.4 Configuration

The purpose of configuration is to move the pan-and-tilt (P/T) so that the index points are hit to reset the counters that count the pulses coming from the position encoders.

Upon power up or reset, the firmware goes into a configuration cycle. There is a delay to allow the operator to remove his or her hands before the drive starts moving.

After the delay, the lens zooms wide and focuses far. Then it zooms tele and focuses near. Then the Intercept tilts down and pans left. Then it tilts up and pans right. During configuration the receiver will not accept any commands.

4.5 Reset command

The reset command resets the receiver. The receiver will start configuring, just as if power had been removed and then reapplied.

4.6 Motion commands

Motion commands are commands that tell the receiver to start or stop a P/T or lens motion (pan, tilt, zoom, focus, iris). More than one motion can be going on at a time. If a motion (or motions) has been started, the receiver will stop it after 15 seconds when the control comes from an RS-422/RS-485 protocol, or 1 second for Coaxitron commands, if no other motion command has been received. This prevents the receiver from being driven forever if communication problems (or other reasons) prevent the receiver from receiving a stop command.

The second video line (preset labels) is cleared. Also, the first video line (zone labels, pattern messages) is cleared if the monitor is not locked, a pattern is not being recorded, and zone scan is not enabled.

4.7 Speed Calculation

The Spectra and Esprit series of integrated units utilize either a PMD or PUD motion control chip to control their motion. The motion control chip receives its control from the CPU chip in response to various motion commands. An explination as to how the motion commands were generated is in Section 6, page 75.

4.8 Speed Ramping

To avoid abrupt speed changes (which could cause clunking noises or even motor stalling), the speeds (angular velocities in degrees/second) are ramped up or down. A command that causes an speed change (such as a motion command or a move to preset) does not set the speed directly. Instead, it sets a desired speed and direction. The ramping task compares the current speed (degrees/second) and direction to the desired speed and direction and calculates a new current speed and direction. This new speed is calculated to keep the angular acceleration (degrees/second/second) approximately constant.

Because the time between ramp calculations can vary (depending on the time taken by other tasks), the time since the last ramp calculation is determined by reading a timer and this time is used to help determine the speed change.

4.9 Presets

Presets can be moved to, set, or cleared.

When a move to preset command is received, the preset position stored for the preset number specified in the command is checked. If the position is not valid, the command is ignored. Otherwise the Intercept moves to the preset pan, tilt, zoom, and focus positions.

A move to preset is started by setting preset in progress. This is done during processing of the move to preset command. Then the preset task takes over. On each pass through the preset task, it reads the current position. For each type of movement (e.g. pan), it determines which direction to move to get to the desired position and starts the move. For pan and tilt, it also calculates the speed at which to move (the closer to the destination, the slower). If it is at the desired position, it stops moving. When all types of movement are at the desired position and have been there for some time, preset in progress is turned off.

Once the preset has been reached, the preset label is displayed on the second video line.

If any command which causes motion is received during a move to preset, the move will be aborted and the new command will start. These commands are: a motion command, or another move to preset command, Also if the move is not completed within 15 seconds, the move is aborted and motion is stopped.

When a set preset command is received, the current pan, tilt, focus, and zoom positions are saved for the preset number specified in the command. For video models (all Spectras and Esprits are "video models"), the monitor is unlocked, and the label for that preset becomes whatever is currently on the second video line.

The clear preset command makes the stored preset for the preset number specified in the command invalid so that it can not be moved to.

4.10 Screen Refreshing

Each time this task is called, 5 characters are written to the character generator that puts characters on the screen. Then a counter is incremented so that the next 5 characters are written the next time the task is called. Refreshing the character generator ensures that even if the generator's memory gets corrupted (due to noise or other causes), the memory will be corrected in a short time and bad characters will not stay on the screen.

4.11 Auxiliary outputs

There is 1 auxiliary output. (Spectra and Esprit have 2 auxiliary outputs.) This is a relay that is opened and closed by software command. (The additional output is an "open collector" type.) Note that there is no momentary command to the dome. If a momentary relay closure is desired, a close relay command must be sent to the receiver, followed by an open relay command.

4.12 Zones

Zones are only meaningful for video models.

When a zone start command is received, the current pan position is saved as the start position for the zone number specified in the command. Also whatever is displayed on the first video line is saved as the zone label. Note that zone scan must be turned off before this command is received or the zone programming will not work correctly.

When a zone end command is received, the monitor is unlocked, and the current pan position is saved as the end position for the zone number specified in the command.

Zones extend from the start point clockwise, looking down from above the dome, to the end point. This means that if a zone start point is set, the Intercept is panned slightly clockwise, and the zone end point is set, the zone will be small. But if the Intercept is panned slightly counterclockwise between the start and end points, the zone will be almost all the way around the pan circle. Also note that the requirement that a zone not cross the "zero position" in the pan circle has been removed.

There are commands to turn zone scan on and off. If zone scan has been turned on, the current pan position is continuously read. If the current position is within a zone, the label for that zone is displayed on the first video line. If the current position is not within any zone, the line is cleared.

If the current position is within more than one zone, the label for the highest-numbered zone will be displayed.

4.13 Pattern

There are two types of pattern processing: record and playback. Pattern processing occurs once each timer tick (14 times a second). If recording, the current command is stored in the EEPROM. If the current command is not one that can be played back, an illegal command is saved (it will be skipped during playback). If playing back, a command is read out of the EEPROM and decoded. If it is an illegal command, it is skipped.

Recording of a pattern starts when a start pattern record command is received. It ends when either an end pattern record command is received, or 60 seconds has elapsed since the start pattern record command was received. (Newer units have more patterns that are longer.) For video versions of the code, the message "`PROGRAMMING PATTERN`" is put on the first video line. When recording stops (either by command or timeout), the message is cleared.

Playing of the recorded pattern starts when a start pattern play command is received. When the end of the recorded pattern is reached, playback starts over again at the beginning of the pattern. This continues until any other command is received. For video versions of the code, the monitor is unlocked. If zones have not been enabled, the message "`RUNNING PATTERN`" is put on the first video line and remains there until playback stops, at which point it is cleared. If zones have been enabled, the "`RUNNING PATTERN`" message will not be shown. Instead, the zone labels will be shown as the pattern moves through the zones. When playback is stopped, the first video line is cleared whether the monitor is locked or not. (In the Spectra and Esprit series the "`RUNNING PATTERN`" is never output.)

4.14 Software Description

Note

All code features are described in the following code descriptions. However for each version of code that is generated, only some of the features will be included.

4.14.1 main program

```
initialize hardware and software (including programming the Xilinx)
enable interrupts (the serial receiver is not enabled yet)
put power up message in screen buffer and on screen
delay
put configuring message in screen buffer
start configuration process
do forever
   commands task
   if no scan in progress
      if motion timer has timed out
         stop motion
         stop any command in progress (preset, scan, pattern)
         reset motion timer
      endif
   else (scan in progress)
      switch scan type
         case configure
            configuration task
         endcase
         case preset
            move to preset task
         endcase
         case auto scan
            auto scan task
         endcase
         case frame scan
            frame scan task
         endcase
         case random scan
            random scan task
         endcase
      endswitch
   endif (scan in progress)
   speed ramp task
   alarms task
```

```
   screen refresh task
   pattern task
   if zones are active
      zone task
   endif
enddo
```

4.14.2 Start configuration process

```
set scan type to config
set configuration state to start
set scan in progress
```

4.14.3 Configuration task

```
Start zoom wide and focus far
Wait for a while
Stop motion
Wait for a while
Save the current zoom and focus values
Start zoom tele and focus near
Wait for a while
Stop motion
Wait for a while
read the zoom value
if new zoom value < saved zoom value
   set zoom reversed
endif
read the focus value
if new focus value < saved focus value
   set focus reversed
endif
Start tilt down and pan left
Wait for a while
Stop motion
Wait for a while
Start tilt up and pan right
Wait for a while
Stop motion
Wait for a while
set configuration done
enable UART receiver so commands can be received
```

Note: During the waits the processor is not just looping in the routine. What is actually happening is that the routine state is saved in a global variable. When waiting, the routine returns to the main loop. On the next pass, the main loop will call this routine again, and the routine will use the state variable to decide where to start executing again.

4.14.4 Commands task (RS485

```
Wait for a start byte to be received
initialize data counter
reset timer
repeat
   if no data byte has been received
      if timeout
         prepare to receive next message (wait for another start
         byte)
      endif
   else
      save data byte
      reset timer
   endif
until all command bytes received
if command is for this receiver address and command format is
proper
   compute checksum
   if checksum is correct
      copy command to decode buffer
      decode command
   endif
prepare to receive next message (wait for another start byte)
```

Note: Waits are implemented in a similar (but not identical) fashion as in the configuration routine. Also this routine is exited and re-entered during the repeat loop while waiting for message bytes.

The difference between the waits in this routine and the configuration routine is that if a command byte has been received, the routine checks if the next byte has been received before returning to the main loop. This allows the routine to "catch up" if passes through the main loop are slow and several bytes have been received between the last pass through this routine and this pass.

4.14.5 Commands task (Coaxitron)

```
Disable vertical interrupt (to stop another Coaxitron command
from interfering)
if number of Coaxitron bits received is not valid
```

```
   enable vertical interrupt
else
   copy command from Coaxitron receive buffer to decode buffer
   set number of Coaxitron bits to 0 (an invalid value)
   enable vertical interrupt
   decode command
endif
```

4.14.6 Decode command

```
(RS485 only)
if not a pattern playback command
   send acknowledgment
endif

(all)
if command is extended command
   if not currently recording pattern or command is end record command
      or command is reset
      if not playback command and currently playing pattern
         stop motion
         stop pattern playing
      endif
      switch on command
         case set preset
            if preset number is in range and no preset move is in
            progress
               do set preset
            endif
         endcase
         case clear preset
            if preset number is in range and no preset move is in
            progress
               do clear preset
            endif
         endcase
         case move to preset
            if preset number is in range
               clear screen
               do start move to preset
            elseif preset number is flip
               clear screen
               do start flip
```

```
      elseif preset number is move to zero point
         clear screen
         do start move to zero point
      endif
   endcase
   case set auxiliary relay
      if relay number is in range
         set bit for relay in lens image
         send lens image
      elseif number is auto scan
         set next scan to auto scan
      elseif number is random scan
         set next scan to random scan
      elseif number is frame scan
         set next scan to frame scan
      endif
   endcase
   case clear auxiliary relay
      if relay number is in range
         clear bit for relay in lens image
         send lens image
      endif
   endcase
   case reset configuration
      restart at beginning of code
   endcase
   case set zone start
      if zone number in range
         do set zone start
      endif
   endcase
   case set zone end
      if zone number in range
         do set zone end
      endif
   endcase
   case write char to screen
      put char in screen buffer
      lock screen
   endcase
   case clear screen
      clear screen buffer
```

```
            unlock screen
         endcase
         case zones on
            set zones active
         endcase
         case zones off
            clear zones active
         endcase
         case start pattern record
            do start pattern record
         endcase
         case end pattern record
            do end pattern record
         endcase
         case start pattern play
            do start pattern play
         endcase
         case set zoom speed
            (Zoom speed is controlled by changing the duty cycle
            of pulses sent to the lens motor)
            if valid speed
               set zoom PWM (pulse width modulation) ratio and saved
               zoom speed
            endif
         endcase
         case set focus speed
            (Focus speed is controlled by changing the duty cycle
            of pulses sent to the lens motor)
            if valid speed
               set focus PWM ratio and saved focus speed
            endif
         endcase
         endswitch
      endif (command is end record etc...)
else (command is PTZ command)
   if command is engage scan
      start scan
   elseif command is terminate scan
      stop motion
      stop scan
   elseif command is auto scan
      set next scan to auto scan
```

```
elseif command is random scan
   set next scan to random scan
elseif command is frame scan
   set next scan to frame scan
else (not special PTZ command )
   if not a stop command
      stop preset
      if command is not from playback and playback in progress
         stop playback
      stop scan
      if screen is not locked and playback not in progress and
      recording not in progress
      if zones not active
         clear text
      else
         clear text except zone line
      endif
   endif
   if the tilt up bit is on and not at up limit stop
      set desired tilt up
      get desired tilt speed from tilt speed table using input tilt
      speed
   elseif the tilt down bit is on and not at down limit stop
      clear desired tilt up
      get desired tilt speed from tilt speed table using input tilt
      speed
   else
      set desired tilt speed to 0 (stop tilt)
   endif
   if the pan left bit is on and not at left limit stop
      clear desired pan right
      if input pan speed is in normal speed range
         get desired pan speed from pan speed table using input
         pan speed
      elseif input pan speed is turbo speed
         set desired pan speed to turbo pan speed
         set desired tilt speed to 0 (stop tilt)
      endif
   elseif the pan right bit is on and not at right limit stop
      set desired pan right
      if input pan speed is in normal speed range
         get desired pan speed from pan speed table using input
```

```
            pan speed
         elseif input pan speed is turbo speed
            set desired pan speed to turbo pan speed
            set desired tilt speed to 0 (stop tilt)
         endif
      else
         set desired pan speed to 0 (stop pan)
      endif
      if the zoom tele bit is on
         clear wide bit and set tele bit in lens image
         send lens image
      elseif the zoom wide bit is on
         clear tele bit and set wide bit in lens image
         send lens image
      else
         clear tele bit and wide bit in lens image
         send lens image
      endif
      if the focus far bit is on
         clear near bit and set far bit in lens image
         send lens image
      elseif the focus near bit is on
         clear far bit and set near bit in lens image
         send lens image
      else
         clear near bit and far bit in lens image
         send lens image
      endif
      if the iris open bit is on
         clear close bit and set open bit in lens image
         send lens image
      elseif the iris close bit is on
         clear open bit and set close bit in lens image
         send lens image
      else
         clear close bit and open bit in lens image
         send lens image
      endif
   endif (not special PTZ)
endif (PTZ)
```

4.14.7 Set preset

```
Mark the preset to be set invalid (if a power failure occurs
before the new preset is fully written, it will appear invalid)
Write the current position into the new preset
Mark the new preset valid
```

4.14.8 Start move to preset

```
Get the desired preset
if desired preset is valid
   reset motion timer
   set zoom and focus speeds to maximum
   do start move to preset 1
endif
```

4.14.9 Start flip

```
Get current position and put into desired position
set desired pan position 180 degrees away
do start move to preset 1
```

4.14.10 Start move to zero point

```
Set desired position to zero
do start move to preset 1
set lens reversals to max (to disable lens movement)
```

4.14.11 Start move to preset 1

```
(version 3.06 and below)
clear screen text
clear reversal counts
set preset in progress

(version 3.07)
clear screen text
get current position
do ramping calculations (to find where to start ramping pan down)
clear reversal counts
set preset in progress
```

4.14.12 Clear preset

```
   Mark the preset invalid
```

4.14.13 Move to preset task

```
if motion timed out
   reset motion timer
   stop preset in progress
   stop motion
elseif preset in progress
   get current position
   if pan reversal count exceeded
      set desired pan speed to 0 (stop pan)
   else
      (start of version 3.06 and below)
      set error to desired pan - current pan
      if SL (cannot go shorter distance for non-SL because of limit
         stops)
         if error > one-half rotation (going right but left is
            closer)
            subtract one rotation from error (compute error going
               left)
         elseif error < -one-half rotation (going left but right is
            closer)
            add one rotation to error (compute error going right)
         endif
      endif
      if (pan is not stalled)
         if error < -minimum allowed pan difference
            clear desired pan right (go left)
            set error to -error (make it positive)
         elseif error > minimum allowed pan difference
            (desired is right of current by more than a threshold)
            set desired pan right
         else (error less than threshold)
            set error to 0
         endif
         if error > ramp error (far enough away to run at full speed)
            set desired pan speed to full speed
         else
            linearly interpolate desired pan speed between full speed
            and zero
         endif
     endif (pan is not stalled)
     (end of version 3.06 and below)
     (version 3.07)
```

```
  do find preset direction (direction pan should move)
  if direction has changed
     do ramping calculations
  endif
  if (pan has not stalled)
     if pan error > error at which to start ramping down
        set desired pan speed to preset speed
     elseif error > 1st step error
        set desired pan speed to 1st step speed (speed at end of
           ramp down)
     elseif error > minimum error
        set desired pan speed to 2nd step speed
     else
        set desired pan speed to 0
     endif
  endif (pan has not stalled)
  (end of version 3.07)
endif (pan reversal count not exceeded)
if tilt reversal count exceeded
   set desired tilt speed to 0 (stop tilt)
else
   set error to desired tilt - current tilt
   if error > minimum allowed tilt difference
      set desired tilt up
   elseif error < -minimum allowed tilt difference
      clear desired tilt up (go down)
   else
      set desired tilt speed to 0 (stop tilt)
   endif
endif
if zoom reversal count exceeded
   clear tele and wide in lens image
   send lens image
else
   set error to desired zoom - current zoom
   if error > 0
      if wide in lens image
         increment zoom reversal
         clear wide in lens image
      endif
      set tele in lens image
      send lens image
```

```
      elseif error < 0
         if tele in lens image
            increment zoom reversal
            clear tele in lens image
         endif
         set wide in lens image
         send lens image
      else
         if wide or tele in lens image
            increment zoom reversal
            clear wide and tele in lens image
            send lens image
            reset preset stop timer
         endif
      endif
   endif (zoom reversal count not exceeded)

   focus is similar to zoom

   if no current motion and no desired motion and preset stop timer
   timed out
      stop preset
   endif
endif
```

Notes: Here is a general description of the way this routine determines that the preset point has been reached. There is a reversal counter for each type of motion (pan, tilt, zoom, focus). A reversal counter is incremented if the desired point for that motion has been reached or if the desired point has been passed and the direction of motion must be reversed. If a reversal counter exceeds a limit, motion is stopped for that motion. This prevents excessive "hunting". When the desired point is reached for a type of motion, that motion is stopped and a timer is reset. If there is no motion at all, then all motions have either reached their desired points or their reversal counters have reached their maximum. But the preset can not be considered complete yet because the mechanics may be coasting past the desired point. So the routine waits for the timer mentioned above to expire before calling the move to preset complete. This gives time to see if any overshoot will occur. If it does, then motion will start again (if the reversal count has not been exceeded), and the timer will start over when the desired point has again been reached and motion has been stopped. Note that the reversal counter and preset stop timer are changed in this routine and also in the ramp routine.

4.14.14 Stop preset

```
set no preset in progress
```

4.14.15 Speed ramp task

```
(also see description of speed calculation in calc.doc)

if not panning
   set current pan speed to zero
endif
if not tilting
   set current tilt speed to zero
endif
if a tick (1/14 second) has elapsed
   set pan stalled flag if movement expected but no
   movement since last tick
endif
determine the number of msec that have occurred since the
last time this task was entered
clear pan and tilt speed change flags
set pan change value to pan speed change per msec * elapsed
msec (this keeps the acceleration rate approximately constant even
if the time between entries to this task varies)
set maximum limit on pan change value
if pan has stalled
   decrease current pan speed to try to achieve motor lock
else
if pan direction needs to be changed and currently panning
   if current pan speed is non zero (slow down before
   reversing)
      decrease current pan speed but not below zero
   else
      if not stopping
         set pan change flag
         reverse direction and increment pan reversal count
      else
         if currently panning
            set pan change flag
            increment pan reversal count
            reset preset stop timer
            stop panning
         endif
     endif
   endif
elseif speed needs to be increased
   set pan change flag
```

```
   increase current pan speed but not above desired speed
   start pan in desired direction
elseif speed needs to be decreased
   set pan change flag
   decrease current pan speed but not below zero or desired speed
else (speeds are equal)
   if stopping pan
      if currently panning
         set pan change flag
         increment pan reversal count
         reset preset stop timer
         stop panning
      endif
   endif
endif
if stopping tilt
   if tilting
      set tilt change flag
      increment tilt reversal count
      reset preset stop timer
      stop tilt
   endif
else
   if tilt up desired
      if currently tilting down
         increment tilt reversal count
         start tilting up
      endif
   else (tilt down desired)
      if currently tilting up
         increment tilt reversal count
         start tilting down
      end
   endif
endif
if current tilt speed is not equal to desired tilt speed
   set tilt change flag
   set current tilt speed to desired tilt speed
endif
if pan change or tilt change flags are set
   (also see calc.doc)
   calculate minimum product (of high and low values) from
```

```
   current pan speed
   calculate maximum product from current tilt speed
   if minimum product is less than or equal to maximum product
      set pan minimum flag
   else
      clear pan minimum flag
      exchange minimum and maximum products
   endif
   calculate minimum high and low values from minimum product
   calculate maximum high and low values from maximum product
   if high speed will be slowed down by using maximum high (for
   example pan 200 degrees/second, tilt 1.5 degrees/second)
     set maximum high value to minimum low value
   endif
   recalculate minimum low value using maximum high value
   recalculate maximum high value using minimum low value to
   eliminate some rounding errors (for example pan 40 degrees/second,
   tilt 1.5 degrees/second)
   recalculate maximum low value using maximum high value
   truncate maximum low value if it is too high
   put maximum high value in hardware
   if pan minimum flag is set
      put minimum low value in hardware for pan speed
      put maximum low value in hardware for tilt speed
   else
      put maximum low value in hardware for pan speed
      put minimum low value in hardware for tilt speed
   endif
endif (change flags were set)
```

4.14.16 Start pattern record

```
put programming message in video text buffer line 1
clear video text buffer line 2
set zones inactive
get current position and save it in the pattern start preset
initialize pattern step
set pattern state to record in progress
```

4.14.17 End pattern record

```
if record in progress
   clear programming message in video text buffer
```

```
   write end pattern marker in pattern
   mark pattern as valid
   clear pattern state
endif
```

4.14.18 Start pattern play

```
start pattern state to move to pattern preset in progress
if pattern is valid
   set pattern state to move to pattern preset
   put running message in video text buffer line 1
   clear video text buffer line 2
   set zones inactive
endif
```

4.14.19 Pattern task

```
switch on pattern state
   if record in progress
      if a tick (1/14 second) has elapsed
         if last decoded command is extended command (not PTZ) or
         speed is greater than normal speed (is turbo speed)
            set last decoded command to illegal command
         endif
         save last decoded command at pattern step location
         increment pattern step
         if pattern full
            clear programming message from text buffer
            clear record in progress
            mark pattern valid
         endif
      endif
   endcase
   case move to pattern preset in progress
      if no preset progress (move done)
         set pattern state to play in progress
         initialize pattern step counter
      endif
   endcase
   case play in progress
      if a tick has elapsed
         if at end of pattern
            start pattern play over
```

```
        else
           get command from pattern step location
           if end pattern marker
              start pattern play over
           else
              decode command
              increment pattern step
           endif
        endif
     endif
  endcase
endswitch
```

4.14.20 Set zone start

```
get current pan and tilt position
set zone start and end positions (in EEPROM) to current pan
position (setting the end position equal to the start position
sets the zone invalid)
put text string in EEPROM
```

4.14.21 Set zone end

```
get current pan and tilt position
set zone end position (in EEPROM) to current pan position
```

4.14.22 Zone task

```
get current pan and tilt position
for zone 8 down to zone 1
   if zone does not cross zero point
      if in zone
         exit for loop
      endif
   elseif zone crosses zero point
      if in zone
         exit for loop
      endif
endfor
if current position is in zone
   put zone text in video text buffer line 1
else
   clear video text buffer line 1
endif
```

4.14.23 Start scan

```
switch on next scan
   case auto scan
      stop motion
      stop commands
      set desired pan right
      set pan speed
      set scan in progress
   endcase
   case frame scan
      stop motion
      stop commands
      set desired pan right
      set pan speed
      reset frame scan timer
      set scan in progress
   endcase
   case random scan
      if no scan in progress or scan type is not random
         stop motion
         stop commands
         initialize random number generator
         set random state to start move
         set scan in progress
      endif
   endcase
endswitch
set scan type to next scan
```

4.14.24 Frame scan task

```
if frame scan timer has timed out
   if currently moving
      stop movement
   else (currently stopped)
      set desired pan right
      start movement
   endif
   reset frame scan timer
endif
```

4.14.25 Random scan task

```
switch on random state
   case start move
      get random value
      if random value is odd (use low random value bit for direction)
         set random right
      else
         clear random right
      endif
      shift random value once right
      normalize random value to a time between minimum move time and
      maximum move time
      set desired pan right to random right
      start motion
      set random state to wait move
      reset motion timer
   endcase
   case wait move
      if motion timer is greater than or equal to random value
         stop motion
         get new random value
         normalize random value to a time between minimum move time
         and maximum move time
         reset motion timer
         set random state to wait look
      endif
   endcase
   case wait look
      if motion timer is greater than or equal to random value
         set random state to start move
      endif
   endcase
```

5 Speed Calculations for Intercept

David Micon

From the circuit, then the *pan step rate* (in steps/second) is given by the following equation:

$$\textit{pan step rate} = 3580000/(256 - H)/2/(256 - PL)/2$$

or

$$\textit{pan step rate} = 895000/((256 - H) \times (256 - PL))$$

where H is the value loaded into the high speed register and PL is the value loaded into the low pan speed register. Each step turns the motor 0.18 degree (it is really a half step). Since there is a 1 : 12 gear reduction, the camera will turn 0.015 degree for each step. So the *pan speed*, or PS, (in degrees/second) is:

$$\textit{pan speed} = 895000/((256 - H) \times (256 - PL)) \times 0.015$$

or

$$PS = 13425/((256 - H) \times (256 - PL))$$

The *tilt step rate* is described by the following equation:

$$\textit{tilt step rate} = 3580000/(256 - H)/2/(256 - TL)/2$$

or

$$\textit{tilt step rate} = 895000/((256 - H) \times (256 - TL))$$

where H is the value loaded into the high speed register and TL is the value loaded into the low tilt speed register. Note that the high speed register is shared between the pan and tilt functions. Each step turns the motor 0.18 degree. Since there is a 1 : 7 gear reduction, the camera will tilt 0.0257 degree for each step. So the *tilt speed*, or TS, (in degrees/second) is

$$\textit{tilt speed} = 895000/((256 - H) \times (256 - TL)) \times 0.0257$$

or

$$TS = 23001/((256 - H) \times (256 - TL))$$

What is desired is to compute values for H, PL, and TL given PS and TS with the constraint that H, PL, and TL must be integers from 0 to 254.

Let $H' = 256 - H$, $PL' = 256 - PL$ and $PP(panproduct) = H' \times PL'$. So

[39] `$Header: d:/UnitSpeeds/RCS/CalcInt.inc,v 1.6 2004-11-09 14:23:08-08 Hamilton Exp Hamilton $`

$$PS = 13425/PPandPP = 13425/PS$$

Once PP has been calculated for a given PS, H and L must be determined. Since H and L must be integers from 0 to 254, PP can only be approximated. If H' is made as small as possible, then L' will be as large as possible. Doing this means that small changes in L' will make small changes in the speed, so the approximation to the desired speed will be more accurate. So set

$$H' = PP/256$$

Then if H' is less than 2, set H' to 2 (this is necessary to keep H from being bigger than 254). Then set

$$PL' = PP/H'$$

If PL' is less than 2 or greater than 256, increase H' by 1 and recalculate PL'. Keep doing this until PL' is greater than or equal to 2 and less than or equal to 256.

Then $H = 256 - H'$ and $PL = 256 - PL'$.

For tilt, let $H' = 256 - H$, $TL' = 256 - TL$ and $TP(\textit{tiltproduct}) = H' \times TL'$. So

$$TS = 23001/TPandTP = 23001/TS$$

Since H and H' have already been determined during the pan calculation, only TL needs to be determined.

$$TL' = TP/H'$$

Since H and H' are fixed for pan, TL' may be greater than 256, so use 256. Then $TL = 256 - TL'$.

5.1 Preset calculations

This shows the calculations needed to determine the point on the P/T pan circle where the time needed to reverse direction to get to a preset is the same as the time needed to continue on in the same direction to get to the preset.

- $p0$ = initial position (degrees)
- $v0$ = initial speed (degrees/second)
- vf = final speed (degrees/second)
- vm = maximum velocity (degrees/second)
- l = total distance (circumference) (360 degrees)
- a = acceleration (degrees/second^2)

- s = distance (degrees)
- t = time (seconds)

$$s = (vf^2 - v0^2)/2 \times a$$
$$t = (vf - v0)/a$$

In the following calculations, deceleration time at the preset position is ignored because it is the same for both reversing and non-reversing motions.

Also the preset position is considered to be 0.

5.1.1 Reversing

distance after decelerating to a stop and accelerating to vm in the opposite direction (ss).

$$ss = ((-vm)^2 - v0^2)/2 \times -a$$

(velocity and acceleration are negative since the motion and acceleration are towards the origin)
time to decelerate to a stop and accelerate to vm in the opposite direction ($t1r$)

$$t1r = (-vm - v0)/-a$$

distance to the preset point after reversing (sar)

$$sar = p0 + ss$$

time to the preset point after reversing (tar)

$$tar = -sar/-vm = sar/vm$$

(distance is negative since the motion is towards the origin)
total time for reversing (tr)

$$tr = t1r + tar$$

5.1.2 Non-reversing

distance after accelerating to vm (da)

$$da = (vm^2 - v0^2)/2 \times a$$

(velocity and acceleration are positive since the motion and acceleration are away from the origin).

time to accelerate to vm ($t1n$)

$$t1n = (vm - v0)/a$$

distance to the preset point after acceleration (sp)

$$sp = l - p0 - da$$

(note: $l - p0$ is the distance to the preset point before acceleration when continuing in the original direction).

time to the preset point after acceleration (tp)

$$tp = sp/vm$$

total time for non-reversing (tn)

$$tn = t1n + tp$$

5.1.3 Final calculation

Find the point where

$$tn = tr$$

and solve for $p0$ in terms of $v0$. By substituting back and simplifying,

$$p0 = l/2 - v0 \times vm/a$$

This shows the calculations needed to determine when to start slowing down to reach the preset.

- $p0$ = initial position (degrees)
- $v0$ = initial speed (degrees/second)
- $p1$ = position to start slowing down
- $v1$ = velocity when slow down starts
- vm = maximum velocity
- a = acceleration (degrees/second^2)
- s = distance (degrees)
- t = time (seconds)

The preset position is considered to be 0.

Speed up phase:

$$p1 = p0 + ((-v1)^2 - (-v0)^2)/2 \times (-a)$$

(speed and acceleration are negative because they are toward the origin)

Slow down phase:

$$0 = p1 + (0^2 - (-v1)^2)/2 \times a$$

(the first zero is the final position and the second zero is the final speed. the acceleration is positive because it is away from the origin)

Substituting back and solving for $v1$:

$$v1 = (+or-)sqrt(a \times p0 + v0^2/2)$$

Substituting into the equation for $p1$ and simplifying,

$$p1 = p0/2 + v0^2/(4 \times a) = v1^2/2 \times a$$

There are limitations on the use of $p1$. If $v0$ is so high that $p1$ is greater than $p0$, this means that the pan can not be stopped in time to stop at the preset and that it will overshoot. Also $v1$ can not be greater than vm. If $v1$ is greater then vm the pan will accelerate vm, coast at the same speed, and decelerate. In this case, $p1$ is:

$$p1 = vm^2/(2 \times a)$$

since $v1$ will be equal to vm. To check whether the maximum speed will be reached, it is easier to check for $v1^2$ then $v1$.

6 Various PMD calculations

David Micon, 199X
Modified by:
Eric Hamilton, 2004

In this document I have collected two worked examples of the PMD specific calculations used in the original Spectra I. The differences are:

1. In Section 6.2, page 76 the following holds:

   Pan Gear Ratio 1 : 5

   Tilt Gear Ratio 1 : 2

   Motor Step Size 0.9^o

2. In Section 6.3, page 80 the following holds:

   Pan Gear Ratio 1 : 12

   Tilt Gear Ratio 1 : 7

   Motor Step Size 1.8^o

Other values are the same. A program to calculate these values is shown in Section 6.4, page 84.

6.1 Fixed point calculations for speed, acceleration and jerk

Calculations are done in long (32-bit unsigned) or long (32-bit signed) arithmetic. The maximum unsigned value is $2^{32} - 1$ or 4294967295. The maximum signed value is $2^{31} - 1$ or 2147483647.

Distances are specified in degrees. To get finer resolution from fixed point numbers, they are stored as tenths of degrees (just called tenths from here on).

[40] `$Header: d:/UnitSpeeds/RCS/PmdCalcs.inc,v 1.5 2004-12-15 07:10:46-08 Hamilton Exp Hamilton $`

6.2 Example 1

6.2.1 Parameters

The gear ratio from pan motor to camera is 1 : 5, and the gear ratio from tilt motor to camera is 1 : 2.

A full step for each motor is 0.9 degrees (deg) and there are 64 microsteps ($\mu steps$) per full step for each motor.

A "cycle" for the PMD motor driver is 540 microseconds or 5.4E-4 seconds.

6.2.2 Converting *pan distance* from *degrees* to *microsteps*

$$1\ deg\ (camera) \times 5\ (motor)/1\ (camera) = 5\ deg\ (motor)$$

(distance ratio is inverse of gear ratio)

$$5\ deg\ (motor)/(0.9\ deg/step) = 5.555555556\ steps$$

$$5.555555556\ steps \times 64\ \mu steps/step = 355.5555556\ \mu steps$$

so

$$1\ deg\ (camera) = 355.5555556\ \mu steps$$

6.2.2.1 Converting *pan speed* from *degrees/second* to *microsteps/cycle*

$$1\ deg/sec\ (camera) \times 355.5555556\ \mu steps/deg = 355.5555556\ \mu steps/sec$$

$$355.5555556\ \mu steps/sec \times 5.4E-4\ sec/cycle = 0.192\ \mu steps/cycle$$

so

$$1\ deg/sec\ (camera) = 0.192\ \mu steps/cycle$$

6.2.2.2 Converting *pan acceleration* from *degrees/second*^{2} to *microsteps*^{2}

$$1\ deg^2\ (camera) \times 0.192\ \mu steps/cycle/deg/sec = 0.192\ \mu steps/cycle/sec$$

$$0.192\ \mu steps/cycle/sec \times 5.4E-4\ usec/cycle = 1.0368E-4\ \mu steps^2$$

so

$$1\ deg^2\ (camera) = 1.0368E-4\ \mu steps^2$$

6.2.2.3 Converting *pan jerk* from *degrees/second*^{3} to *microsteps*^{3}

$$1\ deg^3\ (camera) \times 1.0368E-4\ \mu steps^2/deg^2 = 1.0368E-4\ \mu steps^2/sec$$

$$1.0368E-4\ \mu steps^2/sec \times 5.4E-4\ sec/cycle = 5.59872E-8\ \mu steps^3$$

so

$$1\ deg^3\ (camera) = 5.59872E-8\ \mu steps^3$$

6.2.3 Converting *tilt distance* from *degrees* to *microsteps*

$$1\ deg\ (camera) \times 2\ (motor)/1\ (camera) = 2\ deg\ (motor)$$

(distance ratio is inverse of gear ratio)

$$2\ deg\ (motor)/(0.9\ deg/step) = 2.222222222\ steps$$

$$2.222222222\ steps \times 64\ \mu steps/step = 142.2222222\ \mu steps$$

so

$$1\ deg\ (camera) = 142.2222222\ \mu steps$$

6.2.3.1 Converting *tilt speed* from *degrees/second* to *microsteps/cycle*

$$1\ deg/sec\ (camera) \times 142.2222222\ \mu steps/deg = 142.2222222\ \mu steps/sec$$

$$142.2222222\ \mu steps/sec \times 5.4E-4\ sec/cycle = 0.0768\ \mu steps/cycle$$

so

$$1\ deg/sec\ (camera) = 0.0768\ steps/cycle$$

6.2.3.2 Converting *tilt acceleration* from *degrees/second*^{2} to *microsteps*^{2}

$$1\ deg^2\ (camera) \times 0.0768\ \mu steps/cycle/deg/sec = 0.0768\ \mu steps/cycle/sec$$

$$0.0768\ \mu steps/cycle/sec \times 5.4E-4\ sec/cycle = 4.1472E-5\ \mu steps^2$$

so

$$1\ deg^2\ (camera) = 4.1472E-5\ usteps^2$$

6.2.3.3 Converting *tilt jerk* from *degrees/second*^{3} to *microsteps*^{3}

$$1\ deg^3\ (camera) \times 4.1472E-5\ \mu steps^2/deg^2 = 4.1472E-5\ \mu steps^2/sec$$

$$4.1472E-5\ \mu steps^2/sec \times 5.4E-4\ sec/cycle = 2.239488E-8\ \mu steps^3$$

so

$$1\ deg^3\ (camera) = 2.239488E-8\ \mu steps^3$$

6.2.4 Speed

Speeds must be converted from *degrees/second* (actually *tenths/second*) to *microsteps/cycle*.

The maximum pan speed is 400 *degrees/second* or 4000 *tenths/second*. Speed is expressed as an unsigned value. To avoid overflow, the maximum value of the conversion constant is 4294967295/-4000 or 1073741 (truncated to integer). (Always truncate to keep the value lower than the actual value to avoid overflow). The conversion constant is 0.19 (see above) if the speed is in units of *degrees/second*. Since the speed is in *tenths/second*, the constant is 0.0192. Now we want to multiply this number by the largest power of two that will keep the product less than 1073741. This will keep the largest number of significant bits. In this case this comes out to 2^{25} and the product is 644245 (truncated to integer).

After multiplying to do the conversion, the value is 2^{25}times greater than the actual value (the binary point is 25 places to the left of where it should be). The PMD chipset expects speeds to be expressed in 16/16 format (16 places to the left of the binary point, 16 places to the right of the binary point). So if we shift the result $(25-16)$ or 9 places to the right, the value will be 2^{16} greater than the actual value. So the binary point will be 16 places to the left of where it should be and the value will be scaled to 16/16 format.

The maximum tilt speed is 200 *degrees/second* or 2000 *tenths/second*. Speed is expressed as an unsigned value. The maximum value of the conversion constant is 4294967295/2000 or 2147483 (truncated to integer). The conversion constant is 0.0768 (see above) when the speed is in units of *degrees/second* or 0.00768 for units of *tenths/second*. Multiply the constant by the largest power of two that will keep the product less than 2147483. The power of two is 2^{28} and the product is 2061584 (truncated to integer). To convert to 16/16 format, the product must be shifted $(28-16)$ or 12 places to the right.

6.2.5 Acceleration

The maximum pan acceleration is 3000 *degrees/second*^{2} or 30000 *tenths/second*^{2}. Acceleration is expressed as a signed value (at least for some PMD profile modes). The maximum value of the conversion constant is 2147483647/30000 or 71582 (truncated to integer). The conversion constant is 1.0368E-4 (see above) when the acceleration is in units of *degrees/second*^{2} or 1.0368E-5 for units of *tenths/second*^{2}. Multiply the constant by the largest power of two that will keep the product less than 71582. The power of two is 2^{32} and the product is 44530 (truncated to integer). To convert to 16/16 format, the product must be shifted $(32 - 16)$ or 16 places to the right.

The maximum tilt acceleration is 3000 $degrees/second^2$ or 30000 $tenths/second^2$. Acceleration is expressed as a signed value (at least for some PMD profile modes). The maximum value of the conversion constant is 2147483647/30000 or 71582 (truncated to integer). The conversion constant is 4.1472E-5 (see above) when the acceleration is in units of $degrees/second^2$ or 4.1472E-6 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 71582. The power of two is 2^{34} and the product is 71248 (truncated to integer). To convert to 16/16 format, the product must be shifted $(34 - 16)$ or 18 places to the right.

6.2.6 Jerk

The maximum pan jerk is 10000 $degrees/second^3$ or 100000 $tenths/second^3$. Jerk is expressed as a signed value. The maximum value of the conversion constant is 2147483647/100000 or 21474 (truncated to integer). The conversion constant is 5.59872E-8 (see above) when the acceleration is in units of $degrees/second^2$ or 5.59872E-9 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 21474. The power of two is 2^{41} and the product is 12311 (truncated to integer). To convert to 0/32 format, the product must be shifted $(41 - 32)$ or 9 places to the right.

The maximum tilt jerk is 10000 $degrees/second^3$ or 100000 $tenths/second^3$. Jerk is expressed as a signed value. The maximum value of the conversion constant is 2147483647/100000 or 21474 (truncated to integer). The conversion constant is 2.239488E-8 (see above) when the acceleration is in units of $degrees/second^2$ or 2.239488E-9 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 21474. The power of two is 2^{43} and the product is 19698 (truncated to integer). To convert to 0/32 format, the product must be shifted $(43 - 32)$ or 11 places to the right.

6.3 Example 2

6.3.1 Parameters

The gear ratio from pan motor to camera is 1 : 12, and the gear ratio from tilt motor to camera is 1 : 7.

A full step for each motor is 1.8 degrees (deg) and there are 64 microsteps ($\mu steps$) per full step for each motor.

A "cycle" for the PMD motor driver is 540 microseconds or 5.4E-4 seconds.

6.3.2 Converting *pan distance* from *degrees* to *microsteps*

$$1\ deg\ (camera) \times 12\ (motor)/1\ (camera) = 12\ deg\ (motor)$$

(distance ratio is inverse of gear ratio)

$$12\ deg\ (motor)/(1.8\ deg/step) = 6.66666667\ steps$$

$$6.666666667\ steps \times 64\ \mu steps/step = 426.6666667\ \mu steps$$

so

$$1\ deg\ (camera) = 426.6666667\ \mu steps$$

6.3.2.1 Converting *pan speed* from *degrees/second* to *microsteps/cycle*

$$1\ deg/sec\ (camera) \times 426.6666667\ \mu steps/\ deg = 426.6666667\ \mu steps/sec$$

$$426.6666667\ \mu steps/sec \times 5.4E-4\ sec/cycle = 0.2304\ \mu steps/cycle$$

so

$$1\ deg/sec\ (camera) = 0.2304\ \mu steps/cycle$$

6.3.2.2 Converting *pan acceleration* from *degrees/second^2* to *microsteps/cycle^2*

$$1\ deg/sec^2\ (camera) \times 0.2304\ \mu steps/cycle/\ deg/sec = 0.2304\ \mu steps/cycle/sec$$

$$0.2304\ \mu steps/cycle/sec \times 5.4E-4\ usec/cycle = 1.24416E-4\ \mu steps/cycle^2$$

so

$$1\ deg/sec^2\ (camera) = 1.24416E-4\ \mu steps/cycle^2$$

6.3.2.3 Converting *pan jerk* from *degrees/second*^{3} to *microsteps/cycle*^{3}

$$1\ deg/sec^3\ (camera) \times 1.24416E-4\ \mu steps/cycle^2/\ deg/sec^2 = 1.24416E-4\ \mu steps/cycle^2/sec$$

$$1.24416E-4\ \mu steps/cycle^2/sec \times 5.4E-4\ sec/cycle = 6.718464E-8\ \mu steps/cycle^3$$

so

$$1\ deg/sec^3\ (camera) = 6.718464E-8\ \mu steps/cycle^3$$

6.3.3 Converting *tilt distance* from *degrees* to *microsteps*

$$1\ deg\ (camera) \times 7(motor)/1(camera) = 7\ deg(motor)$$

(distance ratio is inverse of gear ratio)

$$7\ deg(motor)/(1.8\ deg/step) = 3.888888889\ steps$$

$$3.888888889\ steps \times 64\ \mu steps/step = 248.8888889\ \mu steps$$

so

$$1\ deg\ (camera) = 248.8888889\ \mu steps$$

6.3.3.1 Converting *tilt speed* from *degrees/second* to *microsteps/cycle*

$$1\ deg/sec\ (camera) \times 248.8888889\ \mu steps/\ deg = 248.8888889\ \mu steps/sec$$

$$248.8888889\ \mu steps/sec \times 5.4E-4\ sec/cycle = 0.1344\ \mu steps/cycle$$

so

$$1\ deg/sec\ (camera) = 0.1344\ steps/cycle$$

6.3.3.2 Converting *tilt acceleration* from *degrees/second*^{2} to *microsteps/cycle*^{2}

$$1\ deg/sec^2\ (camera) \times 0.1344\ \mu steps/cycle/\ deg/sec = 0.1344\ \mu steps/cycle/sec$$

$$0.1344\ \mu steps/cycle/sec \times 5.4E-4\ sec/cycle = 7.2576E-5\ \mu steps/cycle^2$$

so

$$1\ deg/sec^2\ (camera) = 7.2576E-5\ \mu steps/cycle^2$$

6.3.3.3 Converting *tilt jerk* from *degrees/second*^{3} to *microsteps/cycle*^{3}

$$1\ deg/sec^3\ (camera) \times 7.2576E-5\ \mu steps/cycle^2/\ deg/sec^2 = 7.2576E-5\ \mu steps/cycle^2/sec$$

$$7.2576E-5\ \mu steps/cycle^2/sec \times 5.4E-4\ sec/cycle = 3.919104E-8\ \mu steps/cycle^3$$

so

$$1\ deg/sec^3\ (camera) = 3.919104E-8\ \mu steps/cycle^3$$

6.3.4 Speed

Speeds must be converted from *degrees/second* (actually *tenths/second*) to *microsteps/cycle*.

The maximum pan speed is 400 *degrees/second* or 4000 *tenths/second*. Speed is expressed as an unsigned value. To avoid overflow, the maximum value of the conversion constant is 4294967295/-4000 or 1073741 (truncated to integer). (Always truncate to keep the value lower than the actual value to avoid overflow). The conversion constant is 0.2304 (see above) if the speed is in units of *degrees/second*. Since the speed is in *tenths/second*, the constant is 0.02304. Now we want to multiply this number by the largest power of two that will keep the product less than 1073741. This will keep the largest number of significant bits. In this case this comes out to 2^{25} and the product is 773094 (truncated to integer).

After multiplying to do the conversion, the value is 2^{25} times greater than the actual value (the binary point is 25 places to the left of where it should be). The PMD chipset expects speeds to be expressed in 16/16 format (16 places to the left of the binary point, 16 places to the right of the binary point). So if we shift the result $(25-16)$ or 9 places to the right, the value will be 2^{16} greater than the actual value. So the binary point will be 16 places to the left of where it should be and the value will be scaled to 16/16 format.

The maximum tilt speed is 200 *degrees/second* or 2000 *tenths/second*. Speed is expressed as an unsigned value. The maximum value of the conversion constant is 4294967295/2000 or 2147483 (truncated to integer). The conversion constant is 0.1344 (see above) when the speed is in units of *degrees/second* or 0.01344 for units of *tenths/second*. Multiply the constant by the largest power of two that will keep the product less than 2147483. The power of two is 2^{27} and the product is 1803886 (truncated to integer). To convert to 16/16 format, the product must be shifted $(27 - 16)$ or 11 places to the right.

6.3.5 Acceleration

The maximum pan acceleration is 2000 $degrees/second^2$ or 20000 $tenths/second^2$. Acceleration is expressed as a signed value (at least for some PMD profile modes). The maximum value of the conversion constant is 2147483647/20000 or 107374 (truncated to integer). The conversion constant is 1.24416E-4 (see above) when the acceleration is in units of $degrees/second^2$ or 1.24416E-5 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 107374. The power of two is 2^{33} and the product is 106872 (truncated to integer). To convert to 16/16 format, the product must be shifted $(33 - 16)$ or 17 places to the right.

The maximum tilt acceleration is 2000 $degrees/second^2$ or 20000 $tenths/second^2$. Acceleration is expressed as a signed value (at least for some PMD profile modes). The maximum value of the conversion constant is 2147483647/20000 or 107374 (truncated to integer). The conversion constant is 7.2576E-5 (see above) when the acceleration is in units of $degrees/second^2$ or 7.2576E-6 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 107374. The power of two is 2^{33} and the product is 62342 (truncated to integer). To convert to 16/16 format, the product must be shifted $(33 - 16)$ or 17 places to the right.

6.3.6 Jerk

The maximum pan jerk is 10000 $degrees/second^3$ or 100000 $tenths/second^3$. Jerk is expressed as a signed value. The maximum value of the conversion constant is 2147483647/100000 or 21474 (truncated to integer). The conversion constant is 6.718464E-8 (see above) when the acceleration is in units of $degrees/second^2$ or 6.718464E-9 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 21474. The power of two is 2^{41} and the product is 14774 (truncated to integer). To convert to 0/32 format, the product must be shifted $(41 - 32)$ or 9 places to the right.

The maximum tilt jerk is 10000 $degrees/second^3$ or 100000 $tenths/second^3$. Jerk is expressed as a signed value. The maximum value of the conversion constant is 2147483647/100000 or 21474 (truncated to integer). The conversion constant is 3.919104E-8 (see above) when the acceleration is in units of $degrees/second^2$ or 3.919104E-9 for units of $tenths/second^2$. Multiply the constant by the largest power of two that will keep the product less than 21474. The power of two is 2^{42} and the product is 17236 (truncated to integer). To convert to 0/32 format, the product must be shifted $(42 - 32)$ or 10 places to the right.

6.4 `mtrcalc.c`

Dave Micon also wrote a short routine that will automatically generate the required header file (`.H`) given the input parameters for a given motor step size, etc.

```
/**********************************************************************/
/*
Calculate PMD conversion factors given the pan and tilt step sizes
and gear ratios.

For derivation of the formulas used, see file "calc.doc" in a
PMD code directory.
*/
/**********************************************************************/
#include <stdio.h>
#include <math.h>

#define MAX_PAN_SPEED   400         /* max pan speed in degrees/second */
#define MAX_PAN_ACCEL  3000         /* max pan accel in degrees/second**2 */
#define MAX_PAN_JERK  10000         /* max pan jerk in degrees/second**3 */
#define MAX_TILT_SPEED  200         /* max tilt pan speed in degrees/second */
#define MAX_TILT_ACCEL 3000         /* max tilt accel in degrees/second**2 */
#define MAX_TILT_JERK 10000         /* max tilt jerk in degrees/second**3 */
#define MAX_DWORD      4294967295UL /* max unsigned 32-bit value */
#define MAX_LONG       2147483647L  /* max signed 32-bit value */
#define DISTANCE_SCALE   10         /* distance scale factor to convert degrees to
tenths */
#define USTEPS_STEP      64         /* # of usteps per step */
#define DEGREES_REV     360         /* degrees per revolution */
#define LOG_2          0.69314718055995 /* log(2) */

typedef unsigned int UINT;
typedef unsigned long DWORD;

/**********************************************************************/
/*
Truncate, not round when calculating multipliers to make sure
overflow never occurs.  Round when calculating microsteps per
revolution to get closest value.
*/
/**********************************************************************/
#pragma warning (disable : 4702) /* unreachable code */
void main(void)
    {
    double cycle_time; /* motor controller cycle time */
    double pan_ratio; /* pan gear ratio */
    double pan_step_size; /* pan motor step size */
    double tilt_ratio; /* tilt gear ratio */
```

```
    double tilt_step_size; /* tilt motor step size */
    double distance_conv; /* distance conversion factor */
    double speed_conv; /* speed conversion factor */
    double accel_conv; /* accel conversion factor */
    double jerk_conv; /* jerk conversion factor */
    double max_shifted_conv; /* max shifted conversion value that will
        not make conversion overflow */
    double usteps_rev; /* usteps per revolution */
    DWORD final_conv; /* final conversion value */
    UINT shift_count; /* shift count for conversion value */
    char Inbuff[128]; /* input buffer */

    printf("mtrcalc v1.00\n");
    for (;;)
        {
       do
        {
        pan_ratio = pan_step_size = tilt_ratio = tilt_step_size = 0;
        printf("Cycle time in usec (540 for PMD, 400 for UMDM): ");
        gets(Inbuff);
        sscanf(Inbuff, "%lf", &cycle_time);
            printf("Pan gear ratio: ");
            gets(Inbuff);
            sscanf(Inbuff, "%lf", &pan_ratio);
            printf("Pan step size (degrees): ");
            gets(Inbuff);
            sscanf(Inbuff, "%lf", &pan_step_size);
            printf("Tilt gear ratio: ");
            gets(Inbuff);
            sscanf(Inbuff, "%lf", &tilt_ratio);
            printf("Tilt step size (degrees): ");
            gets(Inbuff);
            sscanf(Inbuff, "%lf", &tilt_step_size);
            printf("\n");
            }
        while (pan_ratio <= 0 || pan_step_size <= 0 || tilt_ratio <= 0 ||
            tilt_step_size <= 0);

        cycle_time *= 1E-6; /* conv from usec to sec */
        printf("Cycle time =      %12.3E\n", cycle_time);
        printf("Pan gear ratio =  %6.2lf\n", pan_ratio);
        printf("Pan step size =   %6.2lf\n", pan_step_size);
        printf("Tilt gear ratio = %6.2lf\n", tilt_ratio);
        printf("Tilt step size =  %6.2lf\n", tilt_step_size);

        /* calculate floating point conversion factors */
        distance_conv = pan_ratio / (pan_step_size * DISTANCE_SCALE) *
            USTEPS_STEP;
```

```
        speed_conv = distance_conv * cycle_time;
        accel_conv = speed_conv * cycle_time;
        jerk_conv = accel_conv * cycle_time;

        /* calculate fixed point conversion factors and shift counts */
        /* pan distance */
        usteps_rev = (DEGREES_REV * DISTANCE_SCALE) * distance_conv; /*
            usteps in one revolution */
        max_shifted_conv = MAX_DWORD / usteps_rev;
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2); /* log base
            2 of max conversion value */
        final_conv = (DWORD) (distance_conv * ((DWORD) 1 <<
            shift_count)); /* shift true conv factor by max allowable */
        printf("Pan distance conversion multiplier:     %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* pan speed */
        max_shifted_conv = MAX_DWORD / ((double) MAX_PAN_SPEED *
            DISTANCE_SCALE * speed_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2); /* log
            base 2 of max conversion value */
        final_conv = (DWORD) (speed_conv * ((DWORD) 1 <<
            shift_count)); /* shift true conv factor by max allowable */
        shift_count -= 16; /* because PMD uses 16/16 value */
        printf("Pan speed conversion multiplier:        %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* pan accel */
        max_shifted_conv = MAX_LONG / ((double) MAX_PAN_ACCEL *
            DISTANCE_SCALE * accel_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2);
        final_conv = (DWORD) (accel_conv * pow(2, shift_count));
        shift_count -= 16; /* because PMD uses 16/16 value */
        printf("Pan acceleration conversion multiplier: %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* pan jerk */
        max_shifted_conv = MAX_LONG / ((double) MAX_PAN_JERK *
            DISTANCE_SCALE * jerk_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2);
        final_conv = (DWORD) (jerk_conv * pow(2, shift_count));
        shift_count -= 32; /* because PMD uses 0/32 value */
        printf("Pan jerk conversion multiplier:         %10ld, shift "

            "count: %2d\n", final_conv, shift_count);
        /* calculate pan microsteps per revolution */
        final_conv = (DWORD) (usteps_rev + 0.5 /* for rounding */);
        printf("Pan microsteps per revolution: %ld\n", final_conv);
```

```
        /* calculate floating point conversion factors */
        distance_conv = tilt_ratio / (tilt_step_size * DISTANCE_SCALE) *
            USTEPS_STEP;
        speed_conv = distance_conv * cycle_time;
        accel_conv = speed_conv * cycle_time;
        jerk_conv = accel_conv * cycle_time;

        /* calculate fixed point conversion factors and shift counts */
        /* tilt distance */
        usteps_rev = (DEGREES_REV * DISTANCE_SCALE) * distance_conv; /*
            usteps in one revolution */
        max_shifted_conv = MAX_DWORD / usteps_rev;
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2); /* log base
            2 of max conversion value */
        final_conv = (DWORD) (distance_conv * ((DWORD) 1 <<
            shift_count)); /* shift true conv factor by max allowable */
        printf("Tilt distance conversion multiplier:     %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* tilt speed */
        max_shifted_conv = MAX_DWORD / ((double) MAX_TILT_SPEED *
            DISTANCE_SCALE * speed_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2); /* log
            base 2 of max conversion value */
        final_conv = (DWORD) (speed_conv * ((DWORD) 1 <<
            shift_count)); /* shift true conv factor by max allowable */
        shift_count -= 16; /* because PMD uses 16/16 value */
        printf("Tilt speed conversion multiplier:        %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* tilt accel */
        max_shifted_conv = MAX_LONG / ((double) MAX_TILT_ACCEL *
            DISTANCE_SCALE * accel_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2);
        final_conv = (DWORD) (accel_conv * pow(2, shift_count));
        shift_count -= 16; /* because PMD uses 16/16 value */
        printf("Tilt acceleration conversion multiplier: %10ld, shift "
            "count: %2d\n", final_conv, shift_count);

        /* tilt jerk */
        max_shifted_conv = MAX_LONG / ((double) MAX_TILT_JERK *
            DISTANCE_SCALE * jerk_conv);
        shift_count = (UINT) (log(max_shifted_conv) / LOG_2);
        final_conv = (DWORD) (jerk_conv * pow(2, shift_count));
        shift_count -= 32; /* because PMD uses 0/32 value */
        printf("Tilt jerk conversion multiplier:         %10ld, shift "
            "count: %2d\n", final_conv, shift_count);
```

```
        /* calculate tilt microsteps per revolution */
        final_conv = (DWORD) (usteps_rev + 0.5 /* for rounding */);
        printf("Tilt microsteps per revolution: %ld\n", final_conv);
        printf("\n\n");
        } /* for (;;) */
    }
#pragma warning (default: 4702) /* unreachable code */
```

6.5 MC.c

Brad Buce modified `mtrcalc.c` for the ExCite project (project 125). The two areas of changes were to make the default gear ratio what is used on the ExCite and have the routine directly generate a header file, `motor.h`, that may be included in source code[41].

```
/*! \class    mc_c
 *
 * - This file is part of BIOS.
 *   $Workfile: mc.c $
 *   $Archive: /RD/Utils/mc/mc.c $
 *   $Revision: 15 $
 *   $Modtime: 7/14/04 8:08a $
 *   $Author: Bbuce $
 *   \attention Copyright(&copy;) by Pelco, 2000, 2001, 2002, 2003
 * -
 * - Calculate PMD conversion factors given the gear ratios.
 * - For derivation of the formulas used, see file "calc.doc" in a
 *   PMD code directory.
 * - Compiles with VC 6.0, runs on a PC. Does not run on the MCORE.
 */

/*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <time.h>

/*lint -e421 Caution -- function 'gets(char *)' is considered dangerous */

void DoAxis(int axis);

#define DEGREES_REV     360           /*!< Degrees per revolution */
#define DISTANCE_SCALE  100            /*!< Distance scale factor to convert degrees 
to hungrees */
#define LOG_2           0.69314718055995 /*!< log(2) */
#define MAX_DWORD       4294967295uL  /*!< Max unsigned 32-bit value */
#define MAX_LONG        2147483647L   /*!< Max signed 32-bit value */

#define PAN             0             /*!< Indicates pan axis is being used */
#define TILT            1             /*!< Indicates tilt axis is being used */
////////////////////////////////////////////////

#define USTEPS_STEP     64            /*!< # of usteps per step */

```

[41]Previously the values generated by `mtrcalc.c` had to be hand transcribed into a header file.

```
#define CYCLE_TIME      542.72
#define MAX_SPEED       40           /*!< Max speed in degrees/second */
#define MAX_ACCEL       100          /*!< Max accel in degrees/second**2 */
#define MAX_JERK        1000         /*!< Max jerk in degrees/second**3 */
#define DENOMINATOR     360
#define NUMERATOR       6800
#define STEP_SIZE       1.8
////////////////////////////////////////////////////////////////////
double accel_conv;         /*!< Accel conversion factor */
double cycle_time;         /*!< Motor controller cycle time */
double denominator;
double distance_conv;      /*!< Distance conversion factor */
double inverse_distance_conv;      /*!< inverse Distance conversion factor */
double jerk_conv;          /*!< Jerk conversion factor */
double max_shifted_conv;   /*!< Max shifted conversion value that will not make conversion overflow */
double max_accel;
double max_jerk;
double max_speed;
double numerator;
double ratio;              /*!< Gear ratio */
double speed_conv;         /*!< Speed conversion factor */
double inverse_speed_conv;
double step_size;          /*!< Motor step size */
int ustep_step;
int distance_scale;
double usteps_rev;         /*!< Usteps per revolution */
unsigned long final_conv;  /*!< Final conversion value */
unsigned int shift_count;  /*!< Shift count for conversion value */
char Inbuff[128];          /*!< Input buffer */
FILE      *outfile;
time_t     start;
struct tm *startlocal;
/*******************************************************************/
/**
Truncate, not round when calculating multipliers to make sure overflow
never occurs.  Round when calculating microsteps per revolution to get
closest value.
*/
```

```
/***********************************************************************/

int main(void)
{
    outfile = fopen("motor.h","w+");

    time(&start);
    startlocal = localtime(&start);

    fprintf(outfile,"/*! \\class    motor_h\n");
    fprintf(outfile," *\n");
    fprintf(outfile," * - This file is part of MAIN\n");
    fprintf(outfile," *   $Workfile: mc.c $\n");
    fprintf(outfile," *   $Archive: /RD/Utils/mc/mc.c $\n");
    fprintf(outfile," *   $Revision: 15 $\n");
    fprintf(outfile," *   $Modtime: 7/14/04 8:08a $\n");
    fprintf(outfile," *   $Author: Bbuce $\n");
    fprintf(outfile," *   \\attention Copyright(&copy;) by Pelco, 2000, 2001, 2002, 2003\n");
    fprintf(outfile," * -\n");
    fprintf(outfile," * - File auto-created: %s",asctime(startlocal));
    fprintf(outfile," */\n\n");

    // Get cycle time
    cycle_time = CYCLE_TIME;
    printf("Cycle time in usec (%f for PMD, 400 for UMDM): %f\n",CYCLE_TIME,cycle_time);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &cycle_time);
    if (cycle_time == 0.0)
        cycle_time = CYCLE_TIME;

    printf("Cycle time        =  %6.3E\n", cycle_time);
    fprintf(outfile,"/**\n");
    fprintf(outfile," * - Cycle time        =  %6.3E\n",cycle_time);


    /* conv from usec to sec */
    cycle_time *= 1E-6;


// Get usteps per step
    ustep_step = USTEPS_STEP;
    printf("Microsteps per step: %d\n",ustep_step);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &ustep_step);
    if (ustep_step == 0.0)
        ustep_step = USTEPS_STEP;
```

```
printf("Usteps per step   =  %d\n", ustep_step);
    fprintf(outfile," * - Usteps per step   =  %d\n",ustep_step);

// print distance scale
distance_scale = DISTANCE_SCALE;
printf("Distance Scale   =  %d\n", distance_scale);
    fprintf(outfile," * - Distance Scale    =  %d\n",distance_scale);
    fprintf(outfile," */\n");

fprintf(outfile,"#define MOTOR_USTEPS_PER_STEP %d\n",ustep_step);

fprintf(outfile,"#define DISTANCE_SCALE %d\n",distance_scale);


    fprintf(outfile,"#define Global_motor_constants_doxygen\n");

    DoAxis(PAN);
    DoAxis(TILT);

    fclose(outfile);
    exit(EXIT_SUCCESS);
}


void DoAxis(int axis)
{
    char AxisPrefix[5];

    ratio = step_size = 0;

    switch (axis)
    {
    case PAN:
        strcpy(AxisPrefix,"PAN");
        break;

    case TILT:
        strcpy(AxisPrefix,"TILT");
        break;

    default:
        return;
    }


    // Get gear ratio
    numerator = 0;
```

```
    printf("%s Numerator of gear ratio: %d\n",AxisPrefix,NUMERATOR);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &numerator);
    if (numerator == 0)
        numerator = NUMERATOR;

    denominator = 0;
    printf("%s Denominator of gear ratio: %d\n",AxisPrefix,DENOMINATOR);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &denominator);
    if (denominator == 0)
        denominator = DENOMINATOR;

    ratio = (double) numerator / denominator;
    printf("A %s gear ratio of %f is being used\n\n",AxisPrefix,ratio);

    // Get step size of motor
    step_size = 0;
    printf("%s Step size (degrees): %f\n",AxisPrefix,STEP_SIZE);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &step_size);
    if (step_size == 0)
        step_size = STEP_SIZE;

    // Get maximum speed, acceleration and jerk parameters
    max_speed = 0;
    printf("%s Maximum speed (degrees/sec): %d\n",AxisPrefix,MAX_SPEED);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &max_speed);
    if (max_speed == 0)
        max_speed = MAX_SPEED;

    max_accel = 0;
    printf("%s Maximum acceleration (degrees/sec/sec): %d\n",AxisPrefix,MAX_ACCEL);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &max_accel);
    if (max_accel == 0)
        max_accel = MAX_ACCEL;

    max_jerk = 0;
    printf("%s Maximum jerk (degrees/sec/sec/sec): %d\n",AxisPrefix,MAX_JERK);
    gets(Inbuff);
    sscanf(Inbuff, "%lf", &max_jerk);
    if (max_jerk == 0)
        max_jerk = MAX_JERK;

    printf("\n\n");

```

```

    fprintf(outfile,"\n/**\n");

    printf("%s Gear ratio        =  %6.2f\n",AxisPrefix, ratio);
    fprintf(outfile," * - %s Gear ratio        =  %6.2f\n",AxisPrefix, ratio);
    printf("%s Step size         =  %6.2f\n",AxisPrefix, step_size);
    fprintf(outfile," * - %s Step size         =  %6.2f\n",AxisPrefix, step_size);
    printf("%s Max speed         =  %6.2f\n", AxisPrefix,max_speed);
    fprintf(outfile," * - %s Max speed         =  %6.2f\n",AxisPrefix, max_speed);
    printf("%s Max acceleration  =  %6.2f\n",AxisPrefix, max_accel);
    fprintf(outfile," * - %s Max acceleration  =  %6.2f\n",AxisPrefix, max_accel);
    printf("%s Max jerk          =  %6.2f\n",AxisPrefix, max_jerk);
    fprintf(outfile," * - %s Max jerk          =  %6.2f\n",AxisPrefix, max_jerk);
    fprintf(outfile," *\n");

    printf("\n");

    /* calculate floating point conversion factors */
    distance_conv         = ratio / (step_size * DISTANCE_SCALE) * USTEPS_STEP;
inverse_distance_conv = 1/distance_conv;
    speed_conv            = distance_conv * cycle_time;
inverse_speed_conv    = 1/speed_conv;
    accel_conv            = speed_conv * cycle_time;
    jerk_conv             = accel_conv * cycle_time;


    printf("%s Distance conversion factor         =
%8.6E\n",AxisPrefix,distance_conv);
    fprintf(outfile," * - %s Distance conversion factor         =
%8.6E\n",AxisPrefix,distance_conv);

printf("%s Inverse Distance conversion factor =
%8.6E\n",AxisPrefix,inverse_distance_conv);
    fprintf(outfile," * - %s Inverse Distance conversion factor =
%8.6E\n",AxisPrefix,inverse_distance_conv);

    printf("%s Speed conversion factor            =
%8.6E\n",AxisPrefix,speed_conv,speed_conv);
    fprintf(outfile," * - %s Speed conversion factor            =
%8.6E\n",AxisPrefix,speed_conv);

printf("%s Inverse Speed conversion factor    =
%8.6E\n",AxisPrefix,inverse_speed_conv,inverse_speed_conv);
    fprintf(outfile," * - %s Speed conversion factor            =
%8.6E\n",AxisPrefix,inverse_speed_conv);

    printf("%s Acceleration conversion factor     =
%8.6E\n",AxisPrefix,accel_conv,accel_conv);
```

```
    fprintf(outfile," * - %s Acceleration conversion factor     = %8.6E\n",AxisPrefix,accel_conv);
    printf("%s Jerk conversion factor            = %8.6E\n",AxisPrefix,jerk_conv,jerk_conv);
    fprintf(outfile," * - %s Jerk conversion factor           = %8.6E\n",AxisPrefix,jerk_conv);
    printf("\n");
    fprintf(outfile," */\n");
    fprintf(outfile,"#define %s_motor_constants_doxygen\n\n",AxisPrefix);

fprintf(outfile,"#define %s_GEAR_RATIO                      %f\n",AxisPrefix, ratio);
fprintf(outfile,"#define %s_MOTOR_STEP_SIZE                 %f\n",AxisPrefix, step_size);

    /* calculate fixed point conversion factors and shift counts */
////////////////////////////////////////////////////////////////////////////////////////
    /* distance */
    /* usteps in 2 revolution */ // need to use 2 revs for preset error calc to not overflow
    usteps_rev = (DEGREES_REV * DISTANCE_SCALE) * distance_conv;
    max_shifted_conv = MAX_DWORD / (2*usteps_rev);

    /* log base 2 of max conversion value */
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);

    /* shift true conv factor by max allowable */
    final_conv = (unsigned long) (distance_conv * ((unsigned long) 1 << shift_count));
    printf("%s Distance conversion multiplier:      %10ld, shift count: %2d\n",AxisPrefix, final_conv, shift_count);

    fprintf(outfile,"#define %s_DISTANCE_TO_PMD                 %duL\n",AxisPrefix, final_conv);
    fprintf(outfile,"#define %s_DISTANCE_PMD_SHIFT_COUNT        %d\n" ,AxisPrefix, shift_count);
//////////////////////////
/* inverse distance */

    /* usteps in one revolution */
    max_shifted_conv = MAX_DWORD / ((usteps_rev) * inverse_distance_conv);

    /* log base 2 of max conversion value */
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);

    /* shift true conv factor by max allowable */
    final_conv = (unsigned long) (inverse_distance_conv * ((unsigned long) 1 << shift_count));
```

```
    printf("%s Inverse Distance conversion multiplier:     %10ld, shift count:
%2d\n",AxisPrefix, final_conv, shift_count);

    fprintf(outfile,"#define %s_INVERSE_DISTANCE_TO_PMD          %duL\n",AxisPrefix,
final_conv);
    fprintf(outfile,"#define %s_INVERSE_DISTANCE_PMD_SHIFT_COUNT %d\n"
,AxisPrefix, shift_count);
/////////////////////////

    /* Speed */
    max_shifted_conv = MAX_DWORD / ((double) max_speed * DISTANCE_SCALE *
speed_conv);

    /* log  base 2 of max conversion value */
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);

    /* shift true conv factor by max allowable */
    final_conv = (unsigned long) (speed_conv * ((unsigned long) 1 << shift_count));

    /* because PMD uses 16/16 value */
    shift_count -= 16;
    printf("%s Speed conversion multiplier:        %10ld, shift count:
%2d\n",AxisPrefix, final_conv, shift_count);

    fprintf(outfile,"#define %s_SPEED_TO_PMD                     %duL\n",AxisPrefix,
final_conv);
    fprintf(outfile,"#define %s_SPEED_PMD_SHIFT_COUNT            %d\n" ,AxisPrefix,
shift_count);
///////////////////////////////////////////////////

/* inverse Speed */
    max_shifted_conv = MAX_DWORD / (((double) max_speed * DISTANCE_SCALE *
speed_conv)/inverse_speed_conv);


    /* log  base 2 of max conversion value */
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);

    /* shift true conv factor by max allowable */
    final_conv = (unsigned long) (inverse_speed_conv * ((unsigned long) 1 <<
shift_count));

    /* because PMD uses 16/16 value */
    shift_count -= 16;
    printf("%s Inverse Speed conversion multiplier:  %10ld, shift count:
%2d\n",AxisPrefix, final_conv, shift_count);

```

```
    fprintf(outfile,"#define %s_INVERSE_SPEED_TO_PMD            %duL\n",AxisPrefix,
final_conv);
    fprintf(outfile,"#define %s_INVERSE_SPEED_PMD_SHIFT_COUNT    %d\n" ,AxisPrefix,
shift_count);


////////////////////////////////////////////////////
    /* Accel */
    max_shifted_conv = MAX_LONG / ((double) max_accel * DISTANCE_SCALE * accel_conv);
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);
    final_conv = (unsigned long) (accel_conv * pow(2.0, (double) shift_count)); //
HAM 6NOV03 lint fix

    /* because PMD uses 16/16 value */
    shift_count -= 16;
    printf("%s Acceleration conversion multiplier:  %10ld, shift count:
%2d\n",AxisPrefix, final_conv, shift_count);

    fprintf(outfile,"#define %s_ACCEL_TO_PMD                    %dL\n",
AxisPrefix,final_conv);
    fprintf(outfile,"#define %s_ACCEL_PMD_SHIFT_COUNT           %d\n" ,AxisPrefix,
shift_count);

    /* Jerk */
    max_shifted_conv = MAX_LONG / ((double) max_jerk * DISTANCE_SCALE * jerk_conv);
    shift_count = (unsigned int) (log(max_shifted_conv) / LOG_2);
    final_conv = (unsigned long) (jerk_conv * pow(2.0, (double) shift_count)); // HAM
6NOV03 lint fix

    /* because PMD uses 0/32 value */
    shift_count -= 32;
    printf("%s Jerk conversion multiplier:          %10ld, shift count:
%2d\n",AxisPrefix, final_conv, shift_count);

    fprintf(outfile,"#define %s_JERK_TO_PMD                     %dL\n",AxisPrefix,
final_conv);
    fprintf(outfile,"#define %s_JERK_PMD_SHIFT_COUNT            %d\n"
,AxisPrefix, shift_count);

    /* calculate Microsteps per revolution */

if (usteps_rev - ((int) usteps_rev) != 0)
{
printf("WARNING: GEAR RATIO YIELDS NON-INTEGER NUMER OF MICROSTEPS PER
REVOLUTION\n");
fprintf(outfile,"//WARNING: GEAR RATIO YIELDS NON-INTEGER NUMER OF MICROSTEPS PER
REVOLUTION\n");
}
```

```

    /* for rounding */
    final_conv = (unsigned long) (usteps_rev + 0.5 );
    printf("%s Microsteps per revolution:           %ld\n\n",AxisPrefix, final_conv);

    fprintf(outfile,"#define %s_USTEPS_REV                       %dL\n",AxisPrefix,
final_conv);

}
//;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
/**
 * $Log: /RD/Utils/mc/mc.c $
 *
 * 15    7/14/04 2:12p Bbuce
 * fix preset 300+ wrong route err
 *
 * 14    7/01/04 2:13p Bbuce
 *
 * 13    2/17/04 8:11a Bbuce
 * added more parameter defines to remove duplicates from main code
 *
 * 12    2/11/04 10:35a Bbuce
 *
 * 11    2/10/04 3:04p Bbuce
 *
 * 10    2/03/04 3:39p Bbuce
 * added warning if microsteps per revolution is non-integer
 *
 * 9     11/14/03 11:14a Bbuce
 * Fixed bug in distance conversion
 *
 * 8     11/06/03 11:28a Ehamilton
 * Fixed an error in calculating tilt values and made more doxygen
 * changes. Also made some lint fixes.
 *
 * 7     11/05/03 12:05p Ehamilton
 * Make changes to help support doxygen.
 *
 * 6     11/04/03 10:41a Bbuce
 * added tilt to auto gen
 *
 * 5     11/04/03 9:48a Bbuce
 *
 * 4     11/04/03 9:47a Bbuce
 * souresafe keywords
*/
//;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
#define mc_c_log_doxygen
```

6.6 SpdCalc.c

Dave Micon wrote a program to generate the non-linear speed table that is used to control Pelco's pan and tilt units. Its output conforms to the reccomendations of the Joy Stick Report.

```
/*********************************************************************/
/*
Calculate speed tables.
*/
/*********************************************************************/
#include <stdio.h>
#include <math.h>
#define MAX_SPEED 400.0 // maximum output speed in 1/10 deg
#define MIN_SPEED 5.0 // min output speed in 1/10 deg
#define MAX_INPUT_SPEED 63 // max input speed value
#define MIN_INPUT_SPEED 7 /* min input speed value after flat
    minimum speed */
void main(void)
    {
    int i;
    FILE *outfile;
    double multiplier, multiplyvalue;
    double adder, addvalue;
    double value;

    outfile = fopen("temp.tmp", "w");
    adder = (MAX_SPEED - MIN_SPEED) / (MAX_INPUT_SPEED - MIN_INPUT_SPEED + 1);
    multiplier = pow(MAX_SPEED - MIN_SPEED, 1.0 / (MAX_INPUT_SPEED - MIN_INPUT_SPEED
+ 1));
    addvalue = multiplyvalue = MIN_SPEED;
    value = (addvalue + multiplyvalue) / 2.0;
    for (i = 0; i < MIN_INPUT_SPEED; i++)
        {
        printf("%4.0f", value);
        fprintf(outfile, "%4.0f", value);
        printf(",");
        fprintf(outfile, ",");
        if (i % 8 == 7)
            {
            printf("\n");
            fprintf(outfile, "\n");
            }
        }
    addvalue = adder;
    multiplyvalue = multiplier;
    for (i = MIN_INPUT_SPEED; i <= MAX_INPUT_SPEED; i++)
        {
        value = (addvalue + multiplyvalue) / 2.0;
```

```
        printf("%4.0f", value + MIN_SPEED);
        fprintf(outfile, "%4.0f", value + MIN_SPEED);
        if (i != MAX_INPUT_SPEED)
            {
            printf(",");
            fprintf(outfile, ",");
            }
        if (i % 8 == 7)
            {
            printf("\n");
            fprintf(outfile, "\n");
            }
        addvalue += adder;
        multiplyvalue *= multiplier;
        }
    fclose(outfile);
    }
```

APPENDIX A

A Patents

Pelco holds some U.S. patents in controlling stepper motors.

A.1 United States Patent 6,566,839

Title Switched capacitor motor driver

Abstract The present invention is a method and apparatus that allows the torque of an electric motor to be increased allowing the motor to be operated at higher speeds and at higher torque without raising the supply voltage, thus allowing for a wider dynamic range of speed and torque to be realized. This is accomplished by connecting one or more capacitors in series with each motor winding, and selectively activating the capacitors at higher speeds where the frequency of the motor resonates with that of the capacitor(s). Switching a capacitor into the circuit with the motor in a particular frequency range allows higher currents to flow through the motor windings resulting in higher torque.

A.2 United States Patent 6,670,783

Title Method and apparatus for improved motor control through current/velocity correction

Abstract The present invention provides a method and apparatus, including a processor and software incorporating a table that contains sets of pre-determined correction values that are used to supply different amounts of power to an electric stepper motor when the motor is operating at different speeds. Use of the correction values in the table allows power to be supplied to the motor in differing amounts that are approximately the same as the power actually required by the motor at different motor speeds or ranges of speeds.

APPENDIX B

Index