HIRDLS

SW-LOC-295

HIGH RESOLUTION DYNAMICS LIMB SOUNDER

Originator: Lawrence Ames Date: 19 March 1998

Sub-ject/Title: **Mirror Scans and Scan Table Entries**

Description/Summary/Contents:

1. This memo gives some background on the Scan Table, and discusses how the interpolation algorithms will interpret it. Various scan patterns are discussed.

Keywords: Software, Scanner, Scan Table Interpolation

Purpose of this Document: ScanTable Entries
(20 char max.)

Reviewed/Approved by: Date (yy-mm-dd):	_____ originator:	_____ ___ HIRDLS flight s/w manager

Advanced Technology Center CAGE Code 65113
Lockheed Martin Missiles & Space
3251 Hanover Street
Palo Alto, CA 94304-1191
United States of America

The scan table is a record containing information about the desired mirror scan pattern. The exact format (bit pattern, units, parameter order) is described elsewhere: this memo discusses details about the information it contains. After appropriate decoding and unit conversions, each entry provides a desired angular position in both azimuth and elevation, a desired angular velocity in az and el, and the time interval allocated for the mirror to achieve that position and velocity. The Scan Table Interpolation code (see SW-LOC-293 and SW-LOC-294) interprets the scan table entries, and calculates an appropriate position, velocity, and acceleration (in both az and el) for each (roughly millisecond-) cycle of the TEU software (Fig. 1). The control software then calculates the torques required to achieve the desired position and velocity, taking into account flexure force, friction, viscosity, residual errors from previous motions, bench accelerations, etc.

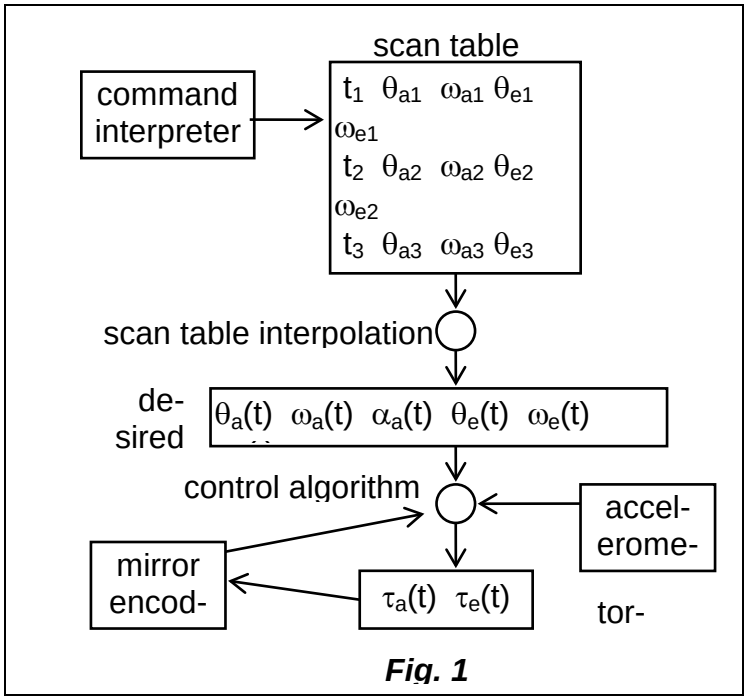

Fig. 1

The scan table does not need a large number of entries: many of the required motions can be described with only a starting entry and an ending entry, and often the ending entry of one motion can serve as the starting entry of the next.

Consider a concrete example (tabulated in Fig. 2, plotted in Fig. 3), with made-up, but illustrative, values: the actual scan table may have additional scans, a different start/end point, different limits, and the safe point located somewhere else. For convenience, time is given in seconds, angles in degrees, and velocities in deg/sec. (The code uses seconds and radians, rad/s, and rad/s^2 internally, but the table can be given in other units, and degrees are easier to visualize.) The first entry says "take one second to reach point (0,0), and be at rest there". Since that is the initial condition in this simulation, the mirror will just sit there for a second; if it had been at some other position, the algorithm output would have been a series of steps that it would move the mirror to there. For actual operation, a longer time interval should be allowed for this initial entry, since the scanner is not necessarily in the initial position upon startup.

```
1,0,0,0,0   } ← acceler-
1,0,0,.5,1  }
4,0,0,4.5,1 } ←const.
2,2,0,4.5,-1} ←transition
4,2,0,.5,-1 } ←const.
2,4,0,.5,1
4,4,0,4.5,1
2,6,0,4.5,-1
4,6,0,.5,-1
2,8,0,.5,1
4,8,0,4.5,1   ← go to "safe"
5,20,0,-1,0 } ←stopped
5,20,0,-1,0 }
5,0,0,0,0   ← return to
```

Fig. 2

The second entry in Fig. 2 tells the az axis to remain stationary, but the elevation axis should **accelerate** to angle 0.5^o in one second, and to have a velocity of +1^o/s at that point. These numbers "work out": a uniform, constant acceleration will result in that position at that time with that velocity ($\theta = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2$, with $\alpha = 1^o/s^2$). As described in SW-LOC-294, the acceleration is not uniform, but rather is the versine function; as discussed in SW-LOC-293, the acceleration during the two half-intervals need not have the same amplitude, although they will when the numbers do work out.

The third entry creates a **uniform scan**: it says that in 4 seconds, the el mirror is to be 4^o more tilted, and that the velocity at the end is to remain 1^o/s. Since the start and ending velocities are the same, and that velocity times the time interval equals the change in angle, the numbers again work out, and the mirror will be commanded in a uniform motion. Obviously, if the ending velocity is different from the start, then there will be accelerations and non-uniform motions. Less obvious: with the same velocity at both ends, if the change in angle is too big for the time interval, then the mirror will be commanded to hurry up for the first half of the interval to make up the distance, and then slow down for the second half to match the required end velocity. Similarly, if the time interval is too long for the change in angle, then the motor will be commanded to decelerate at first to kill time, then accelerate again to match the ending speed.

The fourth entry **transitions** to the second scan. As a user, I don't care how the controllers moves the mirror to the new azimuth position (at 2°): all I want is that it be ready to begin a uniform downward scan. The interpolation algorithm calculates the required az accelerate and decelerate necessary to move the 2° in azimuth and to end up stationary in az, while at the same time decelerating the elevation motion, stopping, and then getting up to speed in the downward direction. In the process, the algorithm describes a curve that smoothly connects the first upward scan with the second, downward one (see upper-left of Fig. 3): the exact shape of the arc is determined by the acceleration profile. Note that care is needed in allocating the time for this transition: too little time will require excessive accelerations; while if too much time is given, the elevation scan will decelerate at a slower rate, and thus the mirror could drift into the limit. As described in SW-LOC-293, the code has an "emergency stop" routine that prevents the mirror from actually hitting the limit, but it jerks the mirror and disrupts the scan pattern: it is far better to check the table entries beforehand for extremes (position limits, as well as peak velocities and peak accelerations—see below).

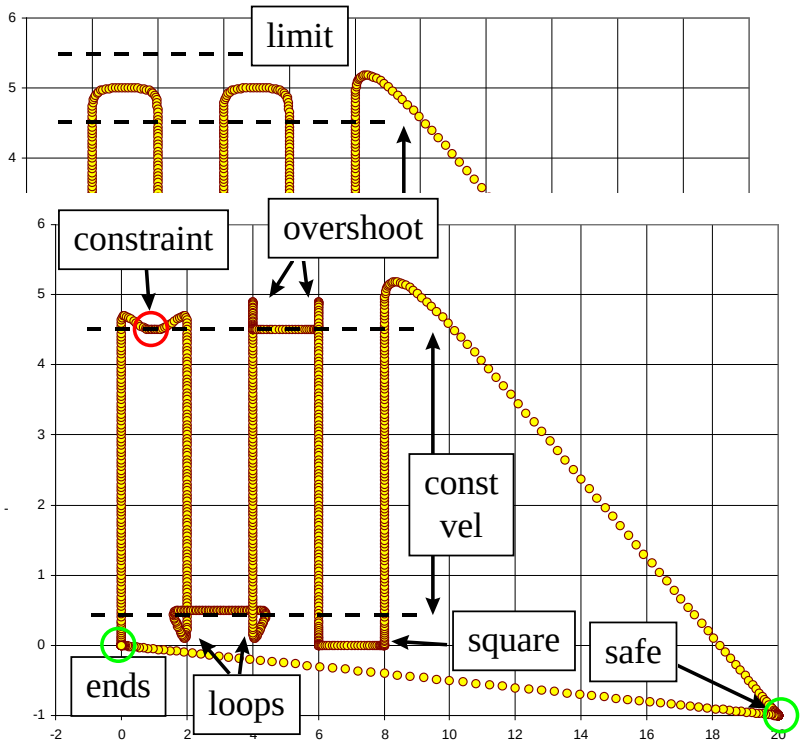

Fig. 4

The next several entries in the table produces the subsequent scans. The third from last entry tells the scanners to move to and stop at the "safe" position, looking at the internal IR reference (simulated here as being at 20,-1). Because the next entry has the same positions, and also have velocities equal to zero, the scanners are **stationary** for the full interval (5 s). Note that simply repeating the positions is not enough: the velocities have to be zero for both entries as well. Otherwise, the scanners will move off position so as to get a "running start" to match the velocity for the next entry.

The last entry in Fig. 2 tells the scanners to return to the initial positions. In normal operation, the software instead would be instructed to repeat the list in the table, in which case this last could be eliminated. Just make sure the first entry has a sufficiently long time interval for the scanner to safely return to the initial position—that is, the last entry could be eliminated if first entry had a 5 s interval. The last entry *is* needed for pretesting the scan table entries, as the testing software uses it to calculate velocities and accelerations for the return segment of the path.

Figure 4 shows alternative methods of transitioning between the constant-velocity scans, with the corresponding scan table entries given in Fig. 5. The first of these methods (shown at the top left) is to **constrain** the transition to have a specific intermediate value. The algorithm will accelerate and decelerate as needed to meet the constraint. A constraint may be useful if specific timing is required for some reason, and the scan would otherwise drift past a limit.

The second method (shown at the bottom left) is to **loop** at the end of the scan. The mirror has a constant velocity in elevation over the full length of the scan, then loops around to get a running start on the azimuth portion of the scan, moves with a constant velocity azimuthally, then loops around again to get a running start on the constant-velocity upward scan. (Note that the loops appear triangular: that is an artifact of the versine acceleration profile, where most of the acceleration occurs at ¼ and ¾ of the way into the time interval; a constant acceleration profile would have created more rounded loops.)

The middle-top of Fig. 4 shows an **overshoot**. While not particularly useful, it shows what the algorithm will do if instructed to end a scan with a given velocity, and then be at rest at that same point at a later time: it will overshoot, come back, and stop. Similarly, if initially at rest at a given point and then instructed to have a given velocity at that same point, the scanner will have to back away some to get a running start, as shown.

The middle-bottom of Fig. 4 shows **square corners**. It is similar in structure to the overshoot/running-start case, except that there is space provided for the scanners to stop and restart. As shown, the azimuth scan in non-uniform in velocity, accelerating for the first half and decelerating for the second. It is possible to specify additional points on that segment so as to create a uniform-velocity portion, if desired.

In addition, there is the arc (not shown in Fig. 4), as discussed earlier. It is the simplest, and probably best, method of transitioning between scans.

A proposed scan table can (and should) be tested with the interpolation algorithm prior to use. While the existing code will need minor modification before it can be used (e.g., to accept entries in the actual format and units, and to use the actual accelerations and limits), the operation should remain unchanged: Run the program “ScanTable” from a Windows prompt. The program will ask for the name of the scan table data file. It will ask if the user wishes an output file that contains the position, velocity, and acceleration for both azimuth and elevation, for each ~millisecond scanner strobe count in the total time in-

```
1,0,0,0,0
1,0,0,.5,1
4,0,0,.5,1                    } ← arc
1,1,1,4.5,0                   ← con-
1,2,0,4.5,-1
4,2,0,.5,-1                   }
2,2,1,.5,0                    } ← loop
2,4,1,.5,0                    }
2,4,0,.5,1                    } ← loop
4,4,0,4.5,1                   }
2,4,0,4.5,0                   } ← over-
2,6,0,4.5,0                   }
2,6,0,4.5,-1                  } ← run start
4,6,0,.5,-1
1,6,0,0,0   ← square   cor-
2,8,0,0,0   ← square   cor-
1,8,0,.5,1
4,8,0,4.5,1
5,20,0,-1,0
5,20,0,-1,0
5,0,0,0,0
```

Fig. 5

terval. The code then runs the table, and reports the extreme angular positions (and the corresponding limits); the maximum speeds (and the speed limits); the maximum accelerations (and the accelerations acceptable for emergency stops); and the peak accelerations outside of an emergency (and the top non-emergency accelerations). As the interpolation algorithm does not check speed and acceleration limits, it is important for the user to check the table entries, and to adjust the entries as appropriate.

~Lawrence Ames.