GBT Memo 231

Trajectory generator with improved small-step behavior

Donald C. Wells*

2004-05-28

Abstract

GBT Memo 203 documented the trajectory generator functions `jmCalcTrajectory()` and `jmPosicastTrajectory()`; the former function is used in the Monitor & Control system of the GBT to minimize vibrations by reducing the 'jerk' of trajectories. Its algorithms have been modified to improve their behavior for small-step trajectories and to implement offset scanning. The revised function prototype for `jmCalcTrajectory()` is documented in this report. Function `jmConcatTrajectory()` has been added to the trajectory generator package[1] to facilitate construction of complex trajectories by concatenation of trajectory segments; this new function is documented here in an appendix.

Revision History

203.0	1999-12-31	GBT Memo 203 distributed	[Wel99]
203.1	2000-10-20	Implemented simulation of small-step trajectory cases	
203.2	2000-12-08	Implemented offset-scan capability	Section 1.1 on page 3
203.3	2000-12-14	Implemented segment concatenation	Appendix A on page 4
203.4	2004-02-02	Implemented linear small-step search	

Contents

*`dwells@nrao.edu` `http://www.cv.nrao.edu/~dwells`
National Radio Astronomy Observatory, 520 Edgemont Road, Charlottesville, VA 22903-2475 USA

[1]The `jm` (jerk minimization) package of C functions is available under GNU Public License at `ftp://fits.cv.nrao.edu/pub/gbt_dwells_jm.tgz` [1141 KB, also see `gbt_dwells_jm.readme`].

1 The new small-step algorithm in `jmCalcTrajectory()`

The first released version of `jmCalcTrajectory()` [Wel99] was installed in the GBT M&C software during the first half of 2000; it was operational in the M&C code during acceptance tests of the GBT main drive servos in August 2000. Steps of various sizes were commanded while output of an accelerometer on the feedarm was monitored. For steps of several degrees J. Brandt reported [Bra00] that the observed vibration levels produced by the jerk-minimizing trajectories were lower than the levels observed when the servos were commanded (by hand) to perform the same steps abruptly. However, for steps of order 0.1 degrees, he reported that

> "One surprising result is that small position changes cause higher magnitude vibrations. I suspect a bug in the algorithm is occurring here when the step is so small, that the acceleration $\sin^2$ shaped pulses begin to narrow enough to cause impulsive application of acceleration forces."

Brandt was certainly correct in his suspicion. This small-step problem was confirmed by simulations soon after it was reported from the field trials.[2] There appear to be three ways in which small steps can lead to impulsive (jerky) trajectories: (1) Function argument `dt` is comparatively large for the GBT, 0.1 s; if Tyler's algorithm [Tyl94] computes an acceleration duration `t1` (or `t3`, deceleration) which is smaller than `dt`, that acceleration will be a step function. (2) Even if an acceleration duration is several times larger than `dt`, it may be undersampled by the piecewise parabolic approximation computed by `jmCalcTrajectory()`, so that the effective trajectory is noisy. (3) The position loops of the GBT AzEl servos have a lowpass cutoff of about 0.3 Hz, so that an acceleration duration which is well-sampled, but which is shorter than this timeconstant (about 3 s), cannot be properly tracked by the PID servos; i.e., significant error will accumulate rapidly in the servo error register, and the servo is likely to jerk the telescope as it tries to zero this error.

The fix for these problems is to modify the algorithm such that all acceleration durations are longer than certain minimum amounts. First, durations should be longer than about 5 times `dt` in order to assure that the piecewise parabolic approximation to the acceleration profile will not be undersampled. For the GBT this will be 0.5 s, and implies that the minimum duration for any trajectory will be about 1 s (acceleration plus deceleration). Second, durations should be comparable (or perhaps slightly greater than) the effective time constant of the PID servo position loops. For the GBT this will be about 3 s, and implies that the minimum duration for any trajectory will be about 6 s.

Soon after the small-step problem was recognized in 2000, the author began to try various fixes for the problem. Several different new algorithms were tried during a three year period, and all were abandoned, for various reasons, after their trials. All of these algorithms shared the feature that `t_servo` was added to the arguments of function `jmCalcTrajectory()` and `jmPosicastTrajectory()`. Also, they all used a minimum acceleration duration which is computed using

```
#define AMSAMPLE 5
t_min = ceil(t_servo[0] / dt) * dt;
t_min = (t_min > AMSAMPLE*dt) ? t_min : AMSAMPLE*dt;
```

The first statement rounds the servo time constant argument up to the next multiple of `dt`, and the second statement assures that profiles will be adequately sampled. The algorithms all attempted to find solutions which satisfy the constraint that both the acceleration and the decelerations times should be longer than `t_min` computed as shown above. Finally, in January 2004 an algorithm which only attempted to satisfy the constraint for *one* of the two times succeeded in generating acceptable trajectories for a set of test problems. This algorithm was further developed during the period January-May 2004, and is being released now, concurrently with publication of this GBT Memo.

The new algorithm tests times t_1 (acceleration) and t_3 (deceleration) to check whether either of them is less than t_{min}. It also computes the changes of position $\frac{1}{2}at_{1,3}^2$ which will occur during the two intervals; if these

[2] This bug is somewhat ironic: the small-step problem was overlooked during the original development in 1999 because of the concentration on solving the large-step case!

are smaller than function argument p_eps (see function prototype below), then the small-step time violation is ignored. If one of the two times is too short, and its motion is significant, the algorithm searches for a reduction of acceleration which will lengthen the time sufficiently. If both of the times are too short and their motions are significant, then the search is conducted for the *larger* of the two times. This design has the deficiency that the other time interval is sometimes still too short, even though the acceleration has been reduced (see Figure 1 on page 5 and Appendix A on the next page for an example). An obvious idea is to search for a solution using the *smaller* of the two times; this approach failed in certain cases, whereas the choice which is described here has succeeded in an extensive set of test cases.[3]

The revised function prototype is:

```
jmError jmCalcTrajectory(                /* returns enum code on error     */
                         jmCalc      mode, /* jmFastest,jmSpecifyTime,..     */
                         jmFunct    funct, /* jmTylerA,jmTylerB,..           */
                         double        dt, /* time step, e.g. 0.1 s          */
                         int         nAxes, /* num axes for p0[],a_max[],etc  */
                         double       p0[], /* initial trajectory             */
                         double       v0[],
                         double       a0[], /*         <--not yet implemented */
                         double      p0s[], /* 'scan' offset wrt trajectory   */
                         double      v0s[],
                         double       pf[], /* final position                 */
                         double       vf[], /* final velocity                 */
                         double       af[], /* final acceleration (jmTylerC)  */
                         double      pfs[], /* 'scan' offset wrt trajectory   */
                         double      vfs[],
                         double        tf, /* time for pf+vf+af target traj  */
                         double    v_max[], /* +/- limits for axes            */
                         double    a_max[],
                         double t_servo[], /* position loop response time     */
                         double    p_eps[], /* negligible position change      */
                         double  *t_total, /* ptr to total trajectory time    */
                         jmPS       **pTS) /* return ptr to private struct    */
```

1.1 The new offset-scan feature in `jmCalcTrajectory`

Ability to generate offset scans was added to the trajectory generator package in December of 2000, and the code was released, accompanied by an unpublished revision of GBT Memo 203 which was called Memo "203.2" (see the Revision History above). This new feature appears as the arguments `p0s`, `v0s`, `pfs` and `vfs` in the function prototype above. These arguments are added to arguments `p0`, `v0`, `pf` and `vf`, respectively. This feature is especially convenient when used in the case of an accelerated target trajectory, as described in Appendix A on the next page.

1.2 Prospects for further improvements for small-step cases

The original `jmCalcTrajectory` implementation solved the problem of finding the fastest trajectory between specified starting and ending conditions. It even solved the problem of finding the fastest trajectory which would osculate to a specified target trajectory. This was an example of an *optimization* problem. The improved algorithm described in this report finds the optimum trajectories subject to a constraint that the

[3]Ideally, we would search for an acceleration reduction which would make *both* times long enough, but that was the approach that failed in the trials during 2000-2003. Even better would be to find *different* acceleration reductions for t_1 and t_3; that is the approach that would be implemented if this code is further developed as discussed in Section 1.2.

acceleration or deceleration time should be longer than a specified minimum time. This is an example of a *constrained* optimization problem. However, the present algorithm can only enforce one such constraint. We really want to simultaneously enforce *two* constaints (on both the acceleration and the deceleration times). During the period 2000-2004, the author tried several approaches in a futile attempt to achieve this result, and ultimately settled for the partial solution presented here.

Algorithms which solve multiple-constraint optimization problems such as this one are available commercially from multiple sources, although not in forms which lend themselves to installation in real-time applications like the GBT Monitor & Control system. Recently the author located a constrained optimization subroutine package which appears to be suitable, and which is in the public domain. It appears to the author that this could be adapted to produce a fully general and fully optimized trajectory generator algorithm.

2 Recommendations for the GBT

The author recommends

- that this new `jm` package be installed in the GBT (with `t_servo` adjusted empirically to minimize excitation of vibrations),
- that function `jmPosicastTrajectory()` be tested with the GBT (to demonstrate *cancellation* of vibrations for appropriate modal period arguments), and
- that the R&D effort discussed in Section 1.2 on the preceding page be authorized and funded in order to make further improvements in the scanning performance of the GBT.

Regarding the second item, the author conjectures that if `jmPosicastTrajectory()` is used to generate trajectories, it may be possible to reduce `t_servo` to well below the actual servo time constant (but not below the sampling constraint of 0.5s for the GBT) without exciting vibrations, because vibrations excited by the servo will be cancelled by the servo. It should also be possible to use the `jmTylerA` mode for the `jmPosicastTrajectory()` accelerations, rather than the `jmTylerB`/`jmTylerC` ($\sin^2$) functions which we now use; this will give a further speedup. These potential improvements may yield more than a factor of two speedup; such a development would be a major improvement for GBT operations if it succeeded.

A Concatenation of offset-scan trajectories (`jmConcatTrajectory`)

This function was originally coded in December of 2000, and has been included with the `jm` code since that time, but it has not been documented in a formal publication. The prototype is:

```
jmError jmConcatTrajectory(           /* returns error code          */
                           jmPS *pPS1, /* output trajectory          */
                           jmPS *pPS2, /* trajectory to be appended */
                           double tol) /* position match tolerance  */
```

The arguments are pointers to two trajectory structures. The second trajectory is appended to the first trajectory. A check is made for position mismatch at the join point, using argument `tol`.

A test problem for this function has been coded, and its output is shown in Figure 1 on the next page. The idea is that we start at position 0.015 with zero velocity, and then move up to osculate to a specified parabolic target trajectory after about 7 s. We then move away from the target trajectory and get a running start for a scan. The scan at constant velocity lasts for 2.4 s, and is centered on the target. We then turn around and do a reverse scan for 2.4 s. Finally, we return to the target trajectory, and osculate to it again.

segment	description	t_{total} [s]	T [s]	t_1 [s]	t_2 [s]	t_3 [s]	am_scale
1	osculate to target trajectory	8.1	8.1	3.42	0.34	3.74	0.0
2	move to start of scan	14.6	6.5	1.91	0.29	3.69	0.0
3	forward scan	17.0	2.4	-0.00	2.40	0.00	1.0
4	reverse direction for 2nd scan	23.6	6.6	3.95	0.06	2.00	0.0
5	reverse scan	26.0	2.4	-0.00	2.40	-0.00	1.0
6	osculate to target trajectory	32.5	6.5	3.70	0.27	1.93	0.0

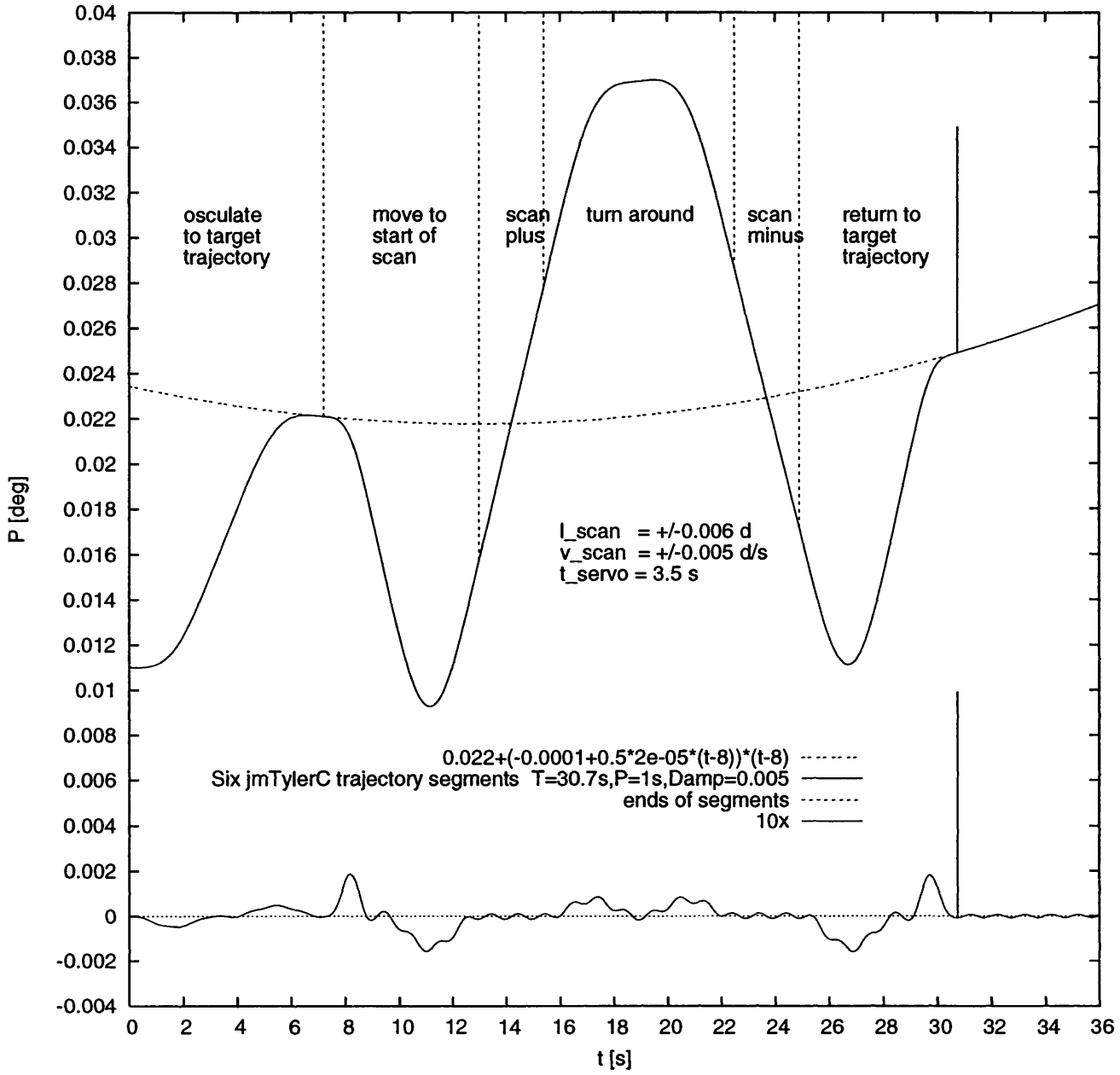

Figure 1: Concatenated trajectory segments relative to accelerating target

This example shows the capability of the `jm` package to generate scan trajectories for the general case of a radio telescope scanning an astronomical target.

The concatenated trajectory was integrated using the mass-on-a-spring approximation, as was done in Memo 203, in order to see the amplitude of vibrations which would be excited by the trajectory. The position error of the system relative to the commanded trajectory is shown as the lower trace of the figure, with 10× exxageration. The negative residual at $t = 1.5$s and the positive residual at $t = 5.5$s are the inertial error of the system (i.e., pointing error while accelerating). Such inertial residuals occur for each of the acceleration and decelleration intervals of the concatenated trajectory. Two of these excursions, at $t = 8$s and $t = 29.5$s, are large and brief. Note that these correspond to the time intervals of order 1.9s in the table at the top of the figure. Also note that oscillations with P=1s start after the first such excursion and are somewhat damped after the second excursion. The vibration amplitudes shown here are less than 0.001 in the figure, but that is really 0.0001. So, if this example is taken as a GBT scan with motions of order 0.03 degree (108 arcsec), then the vibration amplitudes during the scan are less than 0.4 arcsec. This would be an acceptable level for the GBT, even for 100 GHz operation where the beam width is of order 8 arcsec.

The two short time intervals in this simulation illustrate the deficiency of the current algorithm. For example, in segment 6 we have $t_1 = 3.7$s, satisfying the $t_{servo} = 3.5$s constraint, but $t_3 = 1.93$s, which is too short. If these remaining small-step deficiencies of the new algorithm cause difficulty for observations, the author recommends a pragmatic solution: set `t_servo` to a value about 50 percent larger, such as 5 seconds instead of 3.5 seconds. This will be likely to reduce the incidence of small-step problems even further.

References

[Bra00] Joseph J. Brandt. Initial results of the trajectory pre-processor on the GBT. unpublished 4 page limited-distribution memo, with three figures titled 'Position steps using the preprocessor', 'Position steps using the OCU' and 'Position error during steps using the preprocessor', August 2000.

[Tyl94] S. R. Tyler. A trajectory preprocessor for antenna pointing. *[JPL] Telecommunications and Data Acquisition Progress Report 42-118*, April-June:139–159, August 1994. Abstract: "A trajectory-preprocessing algorithm has been devised which matches antenna angular position, velocity, and acceleration to those of a target. This eliminates vibrations of the antenna structure caused by discontinuities in velocity and acceleration commands, and improves antenna-pointing performance by constraining antenna motion to a linear regime..".

[Wel99] Donald C. Wells. Jerk-minimizing trajectory generator in C. GBT Memo 203, NRAO, December 1999. This memo describes the C function `jmCalcTrajectory()`, which computes multiple jerk-minimizing trajectories. This memo also documents the C function `jmPosicastTrajectory()`, which implements the "Posicast" algorithm by convolving multiple trajectories computed by `jmCalcTrajectory()` with pairs of impulses; if the time separations of the impulse pairs is half of the vibrational period(s) *cancelled* (minimized all the way to *zero*). The current version of the `jm` package is available under GNU Public License as file `ftp://fits.cv.nrao.edu/pub/gbt_dwells_jm.tgz`.