JET PROPULSION LABORATORY **INTEROFFICE MEMORANDUM**
MAY 15, 2018 **SCO-18-006**
REVISED SEPTEMBER 1, 2018

TO: Julia Henricks
FROM: Tom Burk
SUBJECT: Enhanced Recon C-Kernels Using Attitude Control Error Telemetry

This memo documents how reconstructed C-Kernels can be enhanced to avoid the 40-microradian quantization granularity which is present in all telemetry-derived reconstructed C-Kernels. This process has been used to improve stellar occultation processing during selected Tour time windows. Cassini attitude control error telemetry can be used to construct a quaternion that is more precise than the raw telemetry quaternion used to form reconstructed C-Kernels. The raw telemetry quaternion is downlinked as a 16-bit signed integer, which is then converted to floating-point on the ground with approximately 40-microradian granularity.

To avoid this granularity problem, the spacecraft attitude can be calculated as follows:

$$\boldsymbol{Q}^{body}{}_{J2000} = \boldsymbol{Q}^{actual\text{-}body}{}_{commanded\text{-}body} \cdot \boldsymbol{Q}^{commanded\text{-}body}{}_{J2000} \qquad \textbf{(Equation 1)}$$

Where $\boldsymbol{Q}^{body}{}_{J2000}$ is the new-formed spacecraft attitude quaternion, which represents a rotation from the J2000 inertial frame to the actual spacecraft body frame. $\boldsymbol{Q}^{actual\text{-}body}{}_{commanded\text{-}body}$ is the rotation from the commanded-body frame to the actual-body frame. This is where the Cassini attitude control error telemetry comes in. The attitude control error can be expressed as a quaternion, if the sign convention of the telemetry is properly accounted for. This quaternion can be used to pre-multiply the commanded attitude quaternion ($\boldsymbol{Q}^{commanded\text{-}body}{}_{J2000}$) to form the desired $\boldsymbol{Q}^{body}{}_{J2000}$ which is then used to create a new C-Kernel. By "pre-multiply", I mean a quaternion multiplication that is analogous to multiplying two direction-cosine matrices: (A-to-B) pre-multiplied by (B-to-C) to form a final (A-to-C) transformation.

The attitude control error is telemetered as three floating-point numbers, expressed in the spacecraft body frame. The telemetry channels for the attitude control error are: A-1180 (X-Axis), A-1181 (Y-Axis, and A-1182 (Z-Axis) and the engineering units for these channels is milliradians, with a range of -3141.6 to +3141.6 milliradians.

This process, and the Cassini telemetry to support it, has been sent to Boris Semenov of the JPL NAIF team. If there are requests to construct these C-Kernels in the future, contact Boris at: Boris.Semenov@jpl.nasa.gov for help.

For this process to work, the commanded attitude quaternion is needed. This quaternion is not available in telemetry. A KPT C-Kernel is used as if it were the commanded attitude of the spacecraft. This assumption is only valid when the spacecraft is quiescent (not turning) and is inertially fixed. By inertially fixed, I mean that "fixed" (constant) IVP vectors are being used in the 7TARGET command to define the spacecraft basebody attitude. This assumption is reasonable when using right ascension and declination (RA/DEC) or other fixed vectors, but is not reasonable when time-varying vectors are used. Examples where this assumption is not valid would be when tracking objects that are moving, like a point on the surface of a celestial body (e.g. Titan or Saturn). Even Earth-pointed targeting is not truly inertially-fixed, so this process cannot generally be used during many science observations. It really only works for inertially-fixed cases. The best way to check if a desired time-window is inertially-fixed is by inspecting the "as flown" sequence 7TARGET command that is "active" during the desired time-window.

The process begins by extracting the attitude control error from telemetry, and changing its sign to be consistent with the convention of quaternion multiplication. The equations to express the attitude control error as a quaternion are:

```
aerr[0] = -1. * aerrx     Where aerrx is A-1180 expressed in milliradians
aerr[1] = -1. * aerry     Where aerrx is A-1181 expressed in milliradians
```

```
   aerr[2] = -1. * aerrz     Where aerrx is A-1182 expressed in milliradians
   aerr_mag = pos_mag(aerr) [pos_mag finds the scalar magnitude of a 3-vector]
```

The scalar aerr_mag is expressed in milliradians. Now find the unit vector of aerr[1-3]:

```
   dunit(aerr, aerr_unit) [dunit finds the unit vector of a 3-vector]
```

The input to dunit is aerr (a 3-vector) and the output is aerr_unit (a 3-vector).

Now construct an attitude control error quaternion:

```
   qcalc_mrad( aerr_mag, aerr_unit, q_actual_commanded)          (Equation 2)
```

Where q_actual_commanded is an output 4-element quaternion explained below.

qcalc_mrad constructs a quaternion using an input scalar magnitude (aerr_mag) and a 3-vector unit (aerr_unit). The quaternion constructed here is a 4-element array with a 3-element vector and a single element scalar that follows the Cassini AACS quaternion convention:

$$ {}^{B}q^{A} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} (\sin\theta/2)a_1 \\ (\sin\theta/2)a_2 \\ (\sin\theta/2)a_3 \\ \cos\theta/2 \end{bmatrix} $$

Cassini AACS quaternion convention

The Cassini AACS quaternion convention is given above, where θ is the Euler rotation angle and a_1, a_2, a_3 are the three components of the "unit rotation vector" (a 3-element unit vector) that define the Euler single-axis rotation from the J2000 frame to the spacecraft body frame. In the C-code below, angle is a scalar input in milliradians, "axis" is the 3-element unit input Euler-axis vector, and q is a 4-element output quaternion.

```
qcalc_mrad(double angle, double axis[], double q[])
double a, b, axis1[3]
a = angle * 1.0e-3 / 2.;
q[3] = cos(a);
b = sin(a);
dunit(axis, axis1); <= Ensure that the input Euler-axis is truly a unit vector
q[0] = axis1[0] * b;
q[1] = axis1[1] * b;
q[2] = axis1[2] * b;
```

Equation (1) is repeated below, where $\boldsymbol{Q}^{commanded\text{-}body}{}_{J2000}$ is the commanded body attitude.

$$\boldsymbol{Q}^{body}{}_{J2000} = \boldsymbol{Q}^{actual\text{-}body}{}_{commanded\text{-}body} \cdot \boldsymbol{Q}^{commanded\text{-}body}{}_{J2000}$$

Where $\boldsymbol{Q}^{commanded\text{-}body}{}_{J2000}$ comes from KPT and $\boldsymbol{Q}^{actual\text{-}body}{}_{commanded\text{-}body}$ is the output of the qcalc-mrad function.

$\boldsymbol{Q}^{body}{}_{J2000}$ is the resulting actual body attitude that can be used to construct a C-Kernel.

The above code is repeated at every 4-second time step. The reason every 4 seconds is used is now explained. Channels A-1180, 1181, and 1182 (attitude control errors) are telemetered every 2 seconds. Reconstructed (as well as KPT) C-Kernels on Cassini are "type 3" C-Kernels in the NAIF vernacular. This means that both the attitude and the spacecraft body rates are needed to construct the C-Kernel. Since spacecraft body rates (channels A-1005, 1006, and 1007) are telemetered every 4 seconds, channels A-1180, 1181, 1182 every two seconds, Equation (1) is computed every 4 seconds, using only the telemetered A-1180, 1181, and 1182 values that match the every-4-second time tag of channels A-1005, 1006, and 1007. So, the attitude control error values that are telemetered "halfway in between" the every-4-second body rates are ignored.

The KPT quaternion, which represents the commanded body attitude is read from the KPT C-Kernel. The NAIF C-Kernel reading function ckgpav_c outputs the KPT quaternion in the form of a direction cosine matrix, `cmat:`

```
   matoq(cmat, quat);      <= Here I convert the KPT cmat back to a quaternion
   dqnorm(quat, quat1);    <= Here I normalize the quaternion
   uqmult(q_actual_commanded, quat1, q_recon);   <= This is Equation 1 above

   dqnorm(q_recon, q_recon_i); <= Here I normalize the resulting quaterion
   qtrpose(q_recon_i, q_i_b);<= Here I take the inverse of the quaternion.  I
need this in Equation 3 below.
```

The NAIF tool that constructs type 3 C-Kernels needs the spacecraft body rates in the J2000 inertial frame, rather than the spacecraft body frame. To convert them to J2000 frame, the following function is used:

```
    uqvtran(body_rates, q_i_b, body_rates_j2000);     (Equation 3)
```

Where body-rates are in rad/s and come in via channels A-1005, 1006, and 1007. q_i_b is the inverse of the KPT quaternion. Just like for direction cosine matrices, the inverse of a quaternion is the "transpose" of the original quaternion. For quaternions, to get the transpose, you just negate the vector component of the quaternion. In the equation below, V is the 3-element vector part of the quaterion and S is the scalar part.

$$ {}^{B}q^{A^*} = {}^{A}q^{B} = q(-\theta,\hat{a}) = q(\theta,-\hat{a}) = \begin{bmatrix} -\bar{V} \\ S \end{bmatrix} $$

In Equation (3), uqvtran is the quaternion analog to multiplying a vector by a direction-cosine matrix to produce an output vector expressed in another coordinate frame. In this case, we are expressing the spacecraft body rates in the J2000 frame. The uqvtran code is:

```
void uqvtran(double v1[], double q[], double v2[])
{
      double a, b, c;
      a = 2.0 * ( q[0] * v1[0] + q[1] * v1[1] + q[2] * v1[2] );
      b = -2.0 * q[3];
      c = q[3] * q[3] - ( q[0] * q[0] + q[1] * q[1] + q[2] * q[2] );
      v2[0] = a * q[0] + b * ( q[1] * v1[2] - q[2] * v1[1] ) + c * v1[0];
      v2[1] = a * q[1] + b * ( q[2] * v1[0] - q[0] * v1[2] ) + c * v1[1];
      v2[2] = a * q[2] + b * ( q[0] * v1[1] - q[1] * v1[0] ) + c * v1[2];
      return;
}
```

The only additional parameter needed to generate a C-Kernel is the time tag of each data point to be used. The time tag needs to be expressed as spacecraft clock (SCLK) encoded into a double-precision representation of the clock count. I have chosen to transform the spacecraft event time (SCET) into the required format. SCET is the format used in KPT processing, and is also available in the channelized telemetry To convert the spacecraft event time (SCET) into SCLK, NAIF functions can be utilized to convert first to ephemeris time, then to a SCLK string, then to SCLK-encoded double-precision. First, the final Cassini SCLK Kernel should be loaded:

```
ldpool_c ( “cas000172.tsc” );
```

Then, a conversion from SCET to ephemeris time is made. “scet” is the input character string in the line below, and ephem_time is the output:

```
   str2et_c ( scet, &ephem_time ) <= Convert SCET to ephemeris time.
```

Then a conversion is made from ephemeris time to a SCLK string:

sce2s_c (sc, ephem_time, lenout, clkstr); <= Convert ephemeris time to a SCLK character string.

For Cassini, the variable “sc” is the integer “-82” and lenout is an integer at least as large as the array holding the SCET string (I make lenout = 22). The output is clkstr, a pointer to a string.

Then the SCLK string is converted to a double-precision representation of the clock count:

`scencd_c ( sc, clkstr, &sclkdp );` **(Equation 4)** <= Express the SCLK character string as a double-precision number. The output is sclkdp, a double-precision representation of the clock count.

Now we want to generate the arrays that will be input to the NAIF routine that constructs the C-Kernel. There is one array for the SCLK time-points, one array for the quaternion in NAIF format, and one array for the body-rates in NAIF format. Each array, in this example, has one dimension for the number of data points, and the array size for that dimenstion is called MAXREC. In my experience, a MAXREC value of 30,000 is sufficient. I define the arrays here:

```
sclk_ck[MAXREC] <=  SCLK-encoded clock count
quat_ck[MAXREC][4] <= The NAIF-formatted quaternion array
avvs[MAXREC][3] <= The NAIF-formatted spacecraft body rate array
```

A loop is created with a time-step of 4 seconds (the time-step of the body-rate telemetry). During each loop, we:

1. Extract the body-rates from the telemetry
2. Extract the attitude control error from the telemetry and process it into quaternion format.
3. Read the KPT C-Kernel to form the commanded attitude quaternion
4. Construct the adjusted quaternion using Equation (1) above.
5. Store the SCLK data, quaternion data, and spacecraft body rates in arrays.

The index for the loop is called “icnt” and starts with a value of zero. It increments by one each loop through.

Below, I expand on Step 5 above.

First, fill the next element of the sclk_ch array:

```
   sclk_ck[icnt] = sclkdp; <= From Equation (4)
```

The NAIF format for quaternions is different than the Cassini AACS standard. To convert from Cassini AACS format to NAIF format, you do the following:

```
   quat_ck[icnt][0] = -1. * q_recon_i[3]; <= The QS scalar becomes the first
element of the quaternion array, and it is expressed as a negative number.
   quat_ck[icnt][1] = q_recon_i[0];  <= Vector becomes the 2nd, 3rd, and 4th elements
   quat_ck[icnt][2] = q_recon_i[1];
   quat_ck[icnt][3] = q_recon_i[2];
```

```
The NAIF format for the spacecraft body rates is:
   avvs[icnt][0] = body_rates_j2000[0];
   avvs[icnt][1] = body_rates_j2000[1];
   avvs[icnt][2] = body_rates_j2000[2];
```

The very first pass through the loop, the C-Kernel “segment” identifiers are initialized. In this example, I’m creating a C-Kernel with only one segment. The identifiers (start, begin_time, and nint) are initialized below (first pass through the loop):

```
      start[0] = sclkdp; <= A double-precision array
      begin_time = sclkdp; <= A double-precision scalar
      nint = 1; <= Segment number 1
```

The final pass through the loop, one additional C-Kernel “segment” identifier is initialized to the final SCLK-encoded spacecraft clock count:

```
   end_time = sclkdp; <= A double-precision scalar
```

After the final pass through the loop, the total number of entries in the array (icnt) is retained and used in the function calls that actually generate the C-Kernel. In this example, I want to generate a C-Kernel called 16115_16115ra. First, I open the file to be created:

```
   ckopn_c ( "16115_16115ra", "16115_16115ra", NCOMCH, &ckhan );
```

```
Then I actually fill the C-Kernel with all the array information that I have
stored:
```

```
   ckw03_c ( ckhan,  begin_time, end_time, INST,   REF,    AVFLAG,
                     SEGID,  icnt,    sclk_ck,  quat_ck,  avvs,   nint, start);
```

```
In the above, ckhan is an integer output of ckopn_c and is used as input into
ckw03_c.  begin_time and end_time are defined above, INST for Cassini is an
integer variable that should be set to -82000, REF is a C string constant
defined as “J2000”.
```

```
AVFLAG is the “angular velocity available” and is defined to be of SPICE
Boolean type set to true:
      #define  AVFLAG         SPICETRUE
```

SEGID is the segment identifier and is defined to be a string set to:

```
      #define  SEGID          "CASSINI RECON TYPE 3 SEGMENT"
```

icnt is the final value of the loop index, sclk_ck, quat_ck, and avvs are the arrays that were calculated during each pass through the loop. nint is the segment identifier set to 1 above, and start is the double-precision array that had its first array value set to the first SCLK value of sclkdp the first pass through the loop.

The resulting high-fidelity C-Kernel should be compared against the KPT and the reconstructed C-Kernels to ensure that the process correctly incorporates the attitude control error telemetry.

As a concluding note, here I explain why this process is really only applicable during periods when the spacecraft is inertially-fixed. When the spacecraft is not inertially-fixed, on-board time-varying IVP vectors are propagated, evaluated, and vector-summed to construct the commanded-body attitude. Each time-varying vector was constructed on the ground based on navigation ephemeris data. These IVP vectors are “fit” to an accuracy of 40 micro-radians. So, at any given time-point, any time-varying IVP vector used in the vector-summation could be “off” the ideal pointing by up to 40 microradians. So, it doesn’t really make sense to use KPT-estimates of the commanded-body attitude during these periods because the on-board commanded body-attitude itself can be “off” by this much, and it is not really practical to do the IVP vector propagation and cubic-spline interpolation on the ground for these cases. That is why we restrict the above process to time-periods where the spacecraft is inertially fixed.

Update on Time-Varying Inertial Targets

KPT is the most easily accessed source for the commanded attitude during the Cassini mission. The standard mode for KPT is to use the loaded ephemeris files, rather than IVP vector propagation, when pointing involves celestial objects (Earth, Sun, Saturn, Titan, etc.) Of course non-celestial objects are sometimes tracked (for example, Radar and Radioscience used Inertial Vector definition files that described their preferred pointing). For these cases, the polynomial vectors constructed by the IVP ground software tool “are” propagated in KPT using polynomial vector propagation. But in KPT, celestial objects always use the loaded ephemeris files. Those same ephemeris files are what the ground IVP Tool uses to build its inertial vectors, but the time-varying inertial vectors produced by IVP are “fit” to 40 microradian accuracy – meaning that over the duration of a vector segment, the IVP-propagated pointing can vary from the “ideal” (i.e. ephemeris) pointing by up to 40 microradians. During a single segment propagation, therefore, the IVP vector fit error will vary over time (see plot below). So, in practice, it is not trivial to know, at any given time, the precise commanded attitude onboard Cassini due to this time-varying nature of the IVP vector fitting error. But for pointing involving celestial objects in KPT, we do know precisely the KPT-commanded attitude – namely, that attitude precisely defined by the loaded ephemeris files. This difference between KPT and onboard commanded attitude makes it problematic to use KPT as the “truth” for the commanded attitude whenever time-varying inertial vectors are being propagated. Fixed vectors do not have this problem, and the KPT-generated commanded attitude does in fact match on onboard commanded attitude, as long as the onboard pointing is using only fixed vectors to define the commanded attitude.

Turn commands also can cause KPT to not match the onboard commanded attitude. In early versions of KPT, the precise timing of “when” a turn starts in KPT could deviate from the onboard turn start time, because KPT did not model all of the steps in the ground system and flight computer. The two key reasons for this are: (1) KPT worked in SCET times, while Cassini used SCLK. (2) KPT used a “hardwired” 2 second delay between turn command times (in the PEF) and the actual start of turn. This delay emulated built-in delays in the CDS and AACS flight computers. In the final version of KPT, created after the Cassini spacecraft plunged into Saturn, new logic was added to correct for these deficiencies. A new input variable called KPT-TURN-START-DELAY was created. Its default value in KPT V14.5 was 3.0 seconds. This value was adjustable by the user (in the KPT Config File), but a single value was used throughout a KPT run.

Several enhancements in KPT V14.5 significantly improved KPT’s ability to correctly model turn starts. The first involved KPT reading the SCLK command time from the PEF (rather than the SCET) and truncating the SCLK fractions of a second (which is what the ground system – specifically SEQTRAN, did). Then convert that SCLK integer back to SCET and use that “corrected SCET” in KPT, instead of the SCETs in the PEF. The second task adjusted the time-delay in KPT between turn command times and the actual start of a turn. This delay emulatd the built-in delays in the CDS and AACS flight computers.

A further task came up during early testing: on-board blocks (OBB) -- for example, mosaics -- issue turn commands slightly differently than the background sequence, so a correction had to be made to deal with this issue. For on-board blocks, the “start” of the block follows the normal SEQTRAN process: SCLK fractions of a section are truncated and the OBB begins at a SCLK integer second. But each command in the OBB, after it is expanded, is issued at a fixed time “relative to the OBB start time” and it is only the OBB start time that undergoes the SCLK truncation process. It was found in testing that some OBBs expand across a window where the SCLK command times cross the SCLK integer threshold. This can lead to KPT turn start command timing errors. To correct for this, an additional set of PEF pre-processing logic was added to KPT V14.5 to more faithfully execute turn commands consistent with the actual spacecraft.

The above changes allow KPT to more faithfully issue turns “in sync” with the actual spacecraft. The improvement in the comparison between flight data and KPT V14.5 were demonstrated during KPT V14.5 User Acceptance Testing. KPT V14.5 was used for all the “final” predicted C-Kernels generated for NAIF archiving into the Planetary Data System.

Categorizing Time-Varying Inertial Targets

If one were to categorize the various situations where using attitude control error telemetry is best suited to improve on the reconstructed C-Kernels, several different categories can be defined. Examples of each are given in the next section.

1. Best: Pointing is truly inertially-fixed throughout the period of interest.
2. Good: Pointing is inertially fixed (as defined via the 7TARGET command), but a series of turns occur during the period of interest
3. Reasonable: Pointing is not inertially-fixed, but the time-varying vectors are changing so slowly over the period the interest, that enhancing the C-Kernel attitude is still better than just using the raw reconstructed C-Kernel
4. OK: Same as Case 3, but a series of turns occur during the period of interest.
5. Poor: Pointing is changing rapidly (for example, near Titan or another moon).
6. Bad: Pointing is not only changing rapidly, but a series of new targets are issued during the period of interest.

Cassini pointing is achieved using pointing commands, turn commands, and IVP vector commands. The commanded pointing attitude begins with a “base” attitude. This attitude is constructed as follows: 1. Specify two spacecraft body vectors which will be used as the primary and secondary body vectors of the base attitude. 2. Specify the inertial vectors which will be used as the primary and secondary inertial vectors of the base attitude. The base attitude is defined as the attitude at which the "primary" body vector is pointed directly at the primary inertial vector, so that they are collinear, while the "secondary" body vector is pointed as close as possible to the secondary inertial vector, subject to the primary vector pointing constraints. When the spacecraft is at the base attitude (with zero offset), the primary body vector is collinear with the primary inertial vector, and all four vectors are coplanar. But in almost all cases, the secondary body vector will *not* be collinear with the secondary inertial vector.

The 7TARGET command is used to define the base attitude. It accepts 4 vector names as arguments. These names specify the: (1) primary body vector; (2) primary inertial vector; (3) secondary body vector; (4) secondary inertial vector.

For example, to specify a base attitude in which the X-Band boresight is pointed at Earth and the spacecraft's -X axis is pointed as close as possible to the Sun, the proper commands are:

```
7TARGET, "XBAND", "EARTH", "NEG_X", "SUN"
             ↑          ↑        ↑       ↑

        Primary    Primary  Secondary  Secondary
        Body       Inertial  Body       Inertial
        Vector     Vector   Vector     Vector
```

It is important to realize that the base attitude will typically change with time due to the propagation of time-varying primary and secondary inertial vectors within the Inertial Vector Table. There are situations where the base attitude will NOT change with time. This occurs when both the primary and secondary inertial vectors are “fixed” vectors -- which means “not” time-varying.

Fixed vectors are defined by explicitly giving the “base” and “head” of the fixed vector, and the three J2000 components of the vector:
7FIXED_VEC, SKY_RA_DEC_100, CASSINI, <x-value>, <y-value>, <z-value>

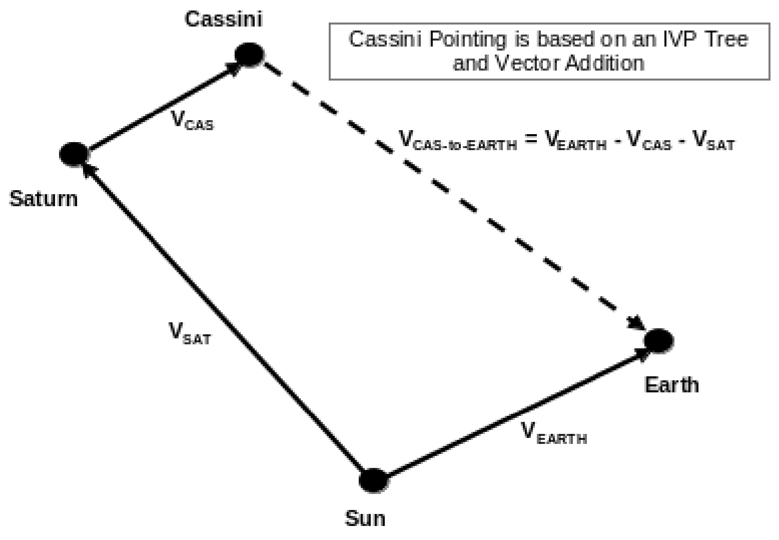
To determine if an inertial vector is time-varying or fixed, you need to inspect the command sequence to see how the inertial vector is defined. Time-varying vectors are of the form:

```
7CONIC_VEC, CASSINI, SATURN, < various arguments >
7POLY_VEC, TITAN, CASSINI, < various arguments >
7ROTATE_CORD, SATURN_FRAME_1, SATURN, < various arguments>
```

There is one more type of fixed vector: a fixed vector attached to a rotating frame:
7ROTATE_VECT, SAT_LAT_LONG_1, SATURN_FRAME_1, < various arguments>

But the rotating frame is typically centered at a celestial body, which is connected to Cassini via a time-varying vector. So, any pointing that uses a rotating-fixed-vector will still be time-varying due to the time-varying nature of the vector that connects Cassini to the rotating frame.

When Cassini is pointing at Earth, there are really 3 time-varying vectors being propagated to achieve the Cassini-to-Earth inertial vector. See below:


Cassini
Cassini Pointing is based on an IVP Tree and Vector Addition
V_{CAS}
$V_{CAS\text{-to-}EARTH} = V_{EARTH} - V_{CAS} - V_{SAT}$
Saturn
V_{SAT}
Earth
V_{EARTH}
Sun

<u>Candidates for using attitude control error telemetry</u>

1. Best: Pointing is truly inertially-fixed throughout the period of interest. Here both inertial vectors are fixed vectors:

   7TARGET, ISS_NAC, SKY_RA_DEC_200, NEG_Z, SKY_RA_DEC_201

Primary Body Vector	Primary Inertial Vector	Secondary Body Vector	Secondary Inertial Vector

   Examples of truly "fixed" inertial vectors are given below:

   SKY_RA_DEC* <= Where "*" means a wildcard where either whitespace or any char string can follow
   SATURN_RA_DEC*
   SKY_NEP
   SATURN_POLE_DIR
   SATURN_N_POLE_DIR
   SATURN_S_POLE_DIR
   TITAN_POLE_DIR
   TITAN_N_POLE_DIR (Sometimes, other moons replace TITAN in this string, like "RHEA_POLE_DIR", or "ENCELADUS_N_POLE_DIR". So perhaps a way of saying this is:
   *_POLE_DIR
   DELTA_H
   DELTA_V
   INERTIAL_YAW

2. Good: Pointing is inertially fixed (as defined via the 7TARGET command), but a series of turns occur during the period of interest
   The examples above also apply here, but there are additional commands that occur during the period of interest, denoting turns occurring:
   7PROFILE, < additional arguments>
   7DELTA_BODY, < additional arguments>
   7DELTA_BASE, < additional arguments >
   7OFFSET, < additional arguments >

   KPT V14.5 does a much better job than previous versions of KPT in terms of starting turns at precisely the same time as Cassini did in flight. Even so, turns can contribute some to differences between KPT-defined commanded attitude and the Cassini on-board commanded attitude. So periods of interest that do contain turns may not be quite as accurate as periods without turns. This is mainly true for time periods where the turn is “in progress”. Once a turn completes, KPT should agree very well with the actual spacecraft. This has been demonstrated in extensive C-Kernel comparison plots of KPT with telemetry-reconstructed C-Kernels throughout the mission at Saturn.

3. Reasonable: Pointing is not inertially-fixed, but the time-varying vectors are changing so slowly over the period the interest, that enhancing the C-Kernel attitude is still better than just using the raw reconstructed C-Kernel.
   7TARGET, XBAND, EARTH, < additional arguments >
   7TARGET, NEG_Z, SUN, < additional arguments >

So, Sun and Earth are close to inertially-fixed, but there is potentially up to a 40-microradian “nearly constant” error due to IVP fitting, for periods of interest less than 12 hours or so. But because the Sun-to-Saturn vector is fit usually better than 40-microradians, this targeting often only introduces less than 10 microradians of (nearly constant) error. As long as the period of interest is not near a celestial-body flyby, and as long as the period of interest is less than 12 hours in duration, KPT probably does a very good job in matching the true commanded attitude.

4. This Case is the same as Case 3, but a series of turns occur during the period of interest.
   7PROFILE, < additional arguments>
   7DELTA_BODY, < additional arguments>
   7DELTA_BASE, < additional arguments >
   7OFFSET, < additional arguments >

   Like for Case 2, turns can contribute some to differences between KPT-defined commanded attitude and the Cassini on-board commanded attitude.

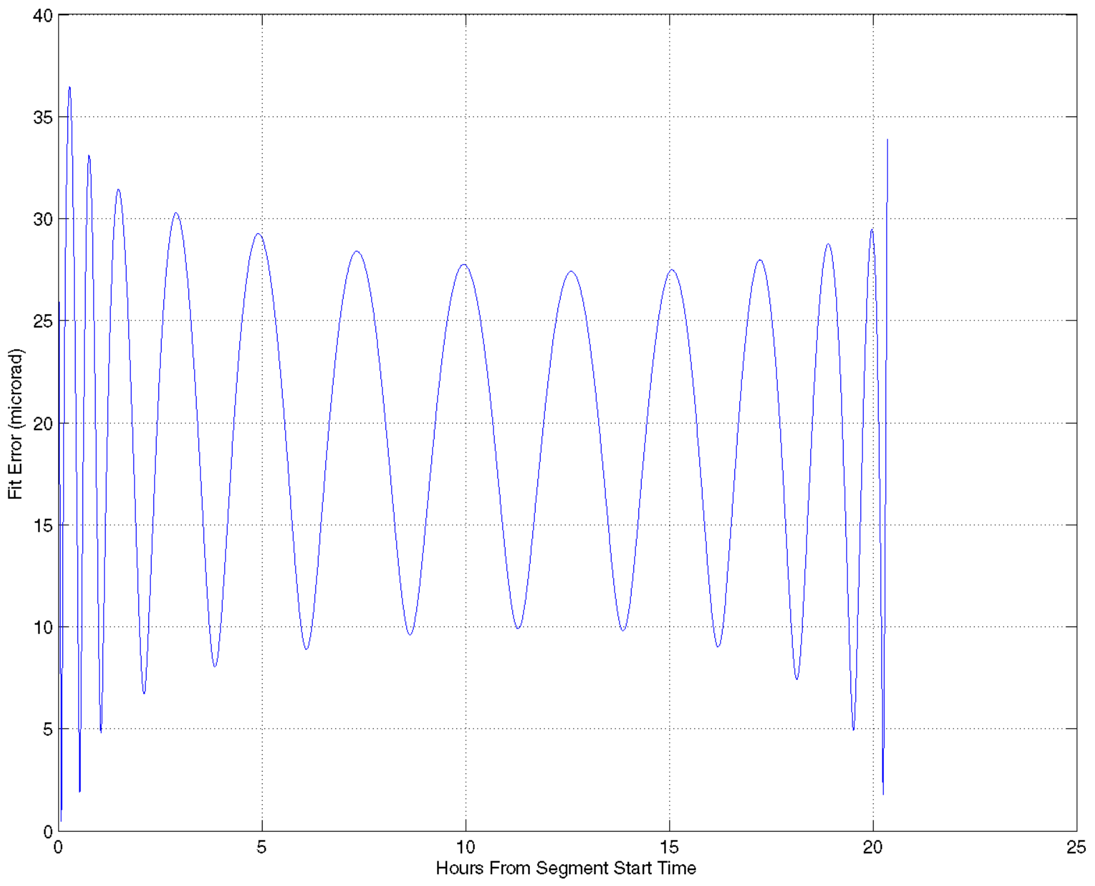
5. Poor: Pointing is changing rapidly (for example, near Titan or other celestial body flybys).
   7TARGET, ISS_NAC, TITAN < additional arguments >
   7TARGET, NEG_Z, TITAN_IVD_RADAR_ < additional arguments >

   Again, although the difference between KPT and Cassini is almost certainly less than 40 microradians, it can reach up to this threshold. Since the telemetry is quantized at roughly 40 microradians, it is unclear, especially near celestial body flybys, if applying the attitude control error technique to KPT-defined commanded attitudes produces “better” pointing when tracking celestial objects when relatively near those objects (as during flybys). This issue is even more problematic when multiple turns occur near celestial body flybys.

6. Bad: Pointing is not only changing rapidly, but a series of new targets are issued during the period of interest.

   T = 0 sec 7PROFILE, < additional arguments>
   T = 5 sec 7TARGET, XBAND, RSSIVD_1, < additional arguments >
   T = 10 sec 7OFFSET, 0, 0, 0
   T = 200 sec 7TARGET, XBAND, TITAN_RSSIVD_2, < additional arguments >
   T = 202 sec 7OFFSET, 0, 0, 0
   T = 300 sec 7TARGET, XBAND, TITAN_RSSIVD_3, < additional arguments >
   T = 302 sec 7OFFSET, 0, 0, 0

Where the RSSIVD vectors are changing rapidly. For example:


Fit Error (microrad)
Hours From Segment Start Time