

SPK

a SOFA-based telescope pointing kernel

Patrick Wallace

Software version 1
Document revision 1.0

2024 November 10

© Copyright 2022-2024 Patrick Wallace. All rights reserved.

Redistribution and use in source (\LaTeX) and ‘compiled’ forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (\LaTeX) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.
2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY THE AUTHOR “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

1	Introduction	1
1.1	Quick Start	1
1.2	What is SPK?	1
2	Pointing and tracking: general principles	2
3	SPK design	3
3.1	Components	3
3.2	Pathways	4
3.3	Data structures	4
3.3.1	Earth orientation parameters	5
3.3.2	Observatory	5
3.3.3	Ambient air conditions	6
3.3.4	Optics	6
3.3.5	Pointing model	6
3.3.6	Hotspot	7
3.3.7	Mount and rotator angles	7
3.3.8	Time	8
3.3.9	Astrometry	9
3.3.10	Target	9
3.4	The three SPK functions	10
3.4.1	<code>spkAstr</code>	10
3.4.2	<code>spkCtar</code>	10
3.4.3	<code>spkVtel</code>	10
3.5	Management of time	11
4	The demonstration programs	12
4.1	Preliminaries	12
4.2	Tracking	17
4.3	The three pathways	21
4.3.1	Target and hotspot to mount angles	21
4.3.2	Mount angles and hotspot to target	22
4.3.3	Mount angles and target to hotspot	22
4.4	Direction of the vertical	23
4.5	The POINT testbed	24
5	Function specifications	26
	<code>spkAstr</code>	26
	<code>spkCtar</code>	28
	<code>spkS2e</code>	29
	<code>spkVtel</code>	32
6	Coordinate conventions	36
6.1	Vectors	36
6.2	Spherical coordinates	36
6.3	Focal plane	36
6.4	Rotator angle	37

7	The operational pointing model	38
7.1	Context	38
7.2	Pointing tests	38
7.3	Modeling	40
7.3.1	Using PMFIT	40
7.4	Implementing the model	43
8	Files	45

1 Introduction

1.1 Quick Start

Compile the demonstrator application `altaz.c`, linked to the SOFA C library¹. All the SPK functions are explicitly included, so for the purposes of the demonstration there is no need for `make` or any precompiled SPK library.

Run the program. The report shows a sequence of tracking position+velocity demands for an altazimuth mount (azimuth, elevation and rotator angle), followed by a series of transformations where any two of

- sky $[\alpha, \delta]$
- mount $[Az, El]$
- focal-plane $[x, y]$

are transformed into the third. This demonstrates all the SPK features needed for the pointing-kernel portion of an altazimuth telescope's control system. Another demonstrator application, `equat.c`, does the same for the case of an equatorially mounted telescope.

1.2 What is SPK?

SPK (simple pointing kernel) is a set of three ANSI C functions that provide low-level support for the pointing and tracking components of a telescope control system (TCS). The positional-astronomy aspects are handled using tools from the IAU SOFA (Standards of Fundamental Astronomy) collection, while the algorithms that predict the telescope mount angles are all from Wallace (2002).²

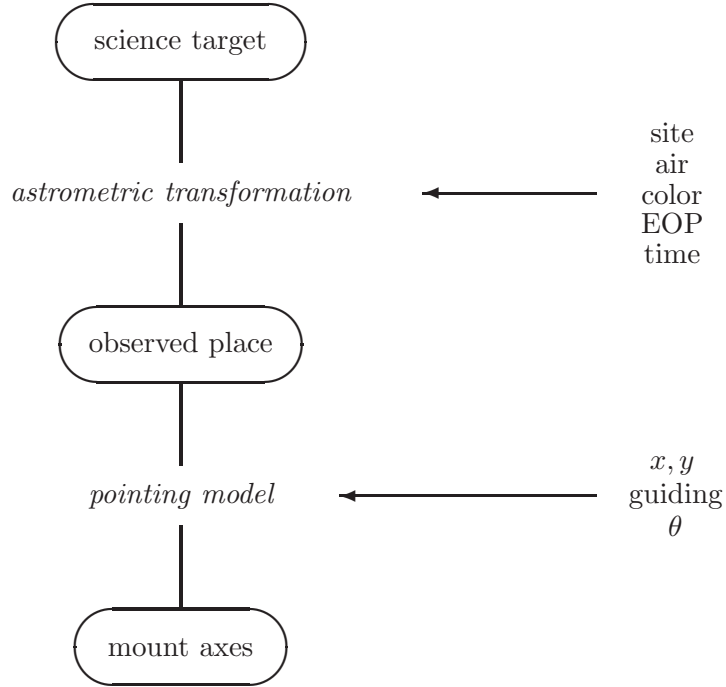
Existing pointing kernels offer a spectrum of trade-offs between feature-richness, efficiency and accuracy. SPK in comparison offers only basic facilities, with flexibility the primary goal. Computational efficiency is secondary, though more than adequate on modern processors. No accuracy compromises are made.

The key element in SPK, and the one that differentiates it from conventional pointing kernel designs, is the concept of the **virtual telescope**. This links three coordinate spaces, namely the sky, the focal plane and the telescope drive demands. All of the basic control modes can be succinctly expressed as deducing one from the other two, rather than being individual set-piece calculations involving spherical trigonometry, small-angle approximations, and so on.

SPK is designed to get the developer off the ground quickly. There are only two key functions to learn about, supplemented by ten data structures. The low-level nature of the API naturally means that compared with other pointing kernel products there are many limitations, for example:

¹See <https://www.iausofa.org>.

²Wallace, P.T., 2002, *A rigorous algorithm for telescope pointing*, Advanced Telescope and Instrumentation Control Software II, edited by Lewis, Hilton, proceedings of the SPIE, 4848, 125. We will refer to this paper as "W02".

Figure 1: **Telescope pointing**

- SPK supports only three celestial coordinate systems, namely ICRS $[\alpha, \delta]$, geocentric apparent $[\alpha, \delta]$ and observed $[Az, El]$;
- Only positions are addressed, not velocities.
- It is limited to self-contained telescope-tube optics, hence no provision for Nasmyth or coudé foci.
- Pointing calibration is left entirely to the TCS developer: see Section 7.

In any TCS based on SPK, these and other required capabilities would be implemented as a higher-level API, with SPK providing the low-level support. Indeed, this minimalist approach may appeal to many developers, who would sooner devise their own tailored ways of (for example) tracking moving targets or managing multiple focal-plane hotspots than be forced to engage with bloated pre-existing mechanisms.

2 Pointing and tracking: general principles

Figure 1 shows how a telescope is pointed at an astronomical target. The problem breaks down into two stages, namely (i) predicting from what direction in the observer's local sky the radiation appears to be coming, followed by (ii) aiming the mount so that the telescope forms an image of the target on a nominated spot in the focal plane.

Because the astrometric transformation in the first stage is time dependent, as a result of Earth rotation in particular, in order to stabilize the target’s image the mount axis angles must change continuously; this process is called “tracking”.

The first stage, namely predicting from what direction the incoming radiation is coming, can be accomplished efficiently and accurately by calling SOFA functions. The second stage, choosing mount axis angles to achieve the desired imaging, is a problem not addressed directly by SOFA; the method that will be described here implements the rigorous formulas set out in paper W02, with SOFA in merely a supporting role.

The present document describes a set of (only) three ANSI C functions that implement Figure 1 and together comprise the rudiments of a “pointing kernel”. Two demonstration `main()` programs, `EQUAT` and `ALTAZ`, are also provided, illustrating different things a real pointing kernel would do.

Note that the TCS design being discussed here assumes that at some fairly rapid rate, say 20-50 Hz, time-stamped demand positions are sent to the mount drives, where servo mechanisms equipped with digital encoders apply the torques needed to achieve the specified locus. Although this is how modern large telescopes work, it is not what amateur mounts do in most cases – typically such a mount deals with the Earth rotation part of Figure 1 itself, with tracking commanded purely in terms of rates, not positions. In these cases the TCS design would be different from the `EQUAT` and `ALTAZ` demonstration programs presented later, although the underlying principles will still apply.

3 SPK design

3.1 Components

The SPK software comprises **a set of data structures** and the following **three functions**:

- `spkAstr` computes the astrometric context, star-independent quantities such as precession-nutation matrices, for a specified UTC. This function can be called to perform either a (computationally expensive) full refresh or alternatively a first-order update that deals only with Earth rotation.
- `spkCtar` accepts an ICRS star catalog entry (epoch J2000.0) and transforms it into astrometric $[\alpha, \delta]$, taking into account space motion and parallax.³
- `spkVtel` implements the algorithms of paper W02. It can be called in a variety of ways, not only to track the celestial target but also to predict image $[x, y]$ for a given $[\alpha, \delta]$ and *vice versa*.

All three functions are re-entrant and thread safe.

Two demonstration `main()` programs are provided (Section 4). Called `equat.c` and `altaz.c`, they are essentially the same thing as each other but configured for an equatorial and an altazimuth mount respectively.

³Although such an $[\alpha, \delta]$ is slowly changing, it is in practice always acceptable to regard it as fixed for the duration of an observation.

3.2 Pathways

At first glance, Figure 1 is simply about pointing a telescope: we know the celestial position, typically an $[\alpha, \delta]$, of the target to be observed, and we want to predict the mount axis angles needed to acquire and track its image. However, this is only the first of three pathways through the diagram:

1. Given the target's $[\alpha, \delta]$ and the desired image $[x, y]$, mount axis angles can be predicted.
2. Given the mount axis angles, the $[\alpha, \delta]$ for a given $[x, y]$ in the focal plane can be calculated.
3. Given the mount axis angles, the $[x, y]$ can be predicted at which the image of a given $[\alpha, \delta]$ will appear.

The algorithms in paper W02, and the function `spkVtel`, perform each of these three transformations rigorously, so that repeated paths back and forth through Figure 1 may accumulate rounding errors but nothing worse. W02 calls this scheme the **virtual telescope**, acknowledging the degree of information hiding that is going on: the TCS developer is insulated from the subtleties and complications of the astrometric transformation and the various misalignments and flexures in the pointing model, and can concentrate on the science objectives.

In the above list of three pathways, the obvious application for the first is, as we already know, pointing and tracking; the second pathway provides a “world coordinate system”, defining how the sky maps onto the focal plane; and the third pathway might be used to predict where an autoguider expects to see the image of a guide star. Another use for the second pathway is to project $[x, y]$ samples onto the celestial coordinate system so that picture orientation can be deduced, and in fact `spkVtel` provides this important capability itself rather than leaving it to the developer.

3.3 Data structures

SPK contains no overall “context”. Instead, the application developer sets out the circumstances for the required calculation by choosing an appropriate set of `spkVtel` arguments, each argument a small data structure. There is complete freedom to assemble whatever selection of these describes the problem to be solved, and updating any one instance will affect only those calls that happen to use it. Different sets of structures could define totally different circumstances, all independently computable without any cross-contamination. For example it is perfectly possible to be dealing with two different targets, one for telescope tracking and the other looking ahead to a forthcoming observation. In other words, every `spkVtel` call is a separate instance of a Virtual Telescope.

There are ten of these data structures, containing a total of 40 data members. The particular choice of granularity keeps argument lists reasonably short without significantly reducing versatility.

Important – the data members are all **public** and must be named explicitly when setting or inquiring values. For example, here is code from the demonstrator `altaz.c` showing how an `spkOBS` structure called `obs` is initialized:

```

iauAf2a ( '-', 5, 41, 54.2, &obs.slon ); /* east longitude */
iauAf2a ( '-', 15, 57, 42.8, &obs.slat ); /* geodetic latitude */
obs.sh = 625.0; /* altitude */
obs.mount = ALTAZ; /* mount type */
obs.pavoid = 0.5 * D2R; /* mount pole avoidance */

```

There are no functions that support individual structures, and consequently no immediate protection against invalid or partial initialization. Descriptions of each of the structures follow.

3.3.1 Earth orientation parameters

The `spkEOP` structure contains three doubles:

- `xp, yp`: the polar motion $[x, y]$ (radians)
- `dut1`: UT1–UTC (seconds)

Values can be obtained from IERS Bulletins. A simple TCS with no pretensions to high accuracy would probably set all of these values (the polar motion at least) to zero.

3.3.2 Observatory

The `spkOBS` structure contains:

- `slon`: site east longitude (ITRF, radians)
- `slat`: site geodetic latitude (ITRF, radians)
- `sh`: site altitude (above WGS84 ellipsoid, meters)
- `mount`: mount type, EQUAT or ALTAZ (see below)
- `pavoid`: closest to mount pole allowed (radians)

All except `mount` are doubles; for `mount` the enumeration `spkMOUNT` offers EQUAT for an equatorial mount or ALTAZ for an altazimuth.⁴

⁴Other gimbal orientations could be supported by choosing ALTAZ and setting the `paw` and `pan` terms in the pointing model (see 3.3.5) to the appropriate large angles.

3.3.3 Ambient air conditions

The `spkAIR` structure contains three `doubles`:

- `p`: pressure (hPa \equiv mB)⁵
- `t`: temperature ($^{\circ}$ C)
- `h`: relative humidity (0–1)

It is a fortunate circumstance that the refraction is almost entirely dependent on these local measurements.

3.3.4 Optics

The `spkOPT` structure contains two `doubles`:

- `fl`: focal length (user units)
- `wl`: wavelength (μ m)

The sole purpose of providing the focal length is to allow the TCS developer to choose in what units to reckon focal-plane coordinates.

The crossover from optical to radio refraction is assumed to occur at `wl`=100 μ m.

3.3.5 Pointing model

To summarize the mechanical imperfections of the telescope and mount, SPK uses a set of seven geometrical terms. The `spkPM` structure contains this seven-term model, supplemented by temporary adjustments to two of them. All data members are radians and `doubles`:

- `pia`: roll ($-\text{HA}$ or $\pi - \text{azimuth}$) index error
- `pib`: pitch (δ or elevation) index error
- `pvd`: droop
- `pca`: telescope/pitch nonperpendicularity
- `pnp`: roll/pitch nonperpendicularity

⁵Pressures reported for a local airfield or read from an electronic weather station are usually what aviators call “QNH”. This is the sea-level pressure and will always hover around 1013 hPa even for a mountaintop site. What SPK needs in order to calculate the refraction is “QFE”, the actual pressure that would be read by a mercury barometer, a much lower figure for high-altitude sites.

- **paw**: roll axis misalignment across meridian
- **pan**: roll axis misalignment along meridian

and

- **ga**: guiding correction applied to **pca**
- **gb**: guiding correction applied to **pib**

It is important to understand that in practice the operational pointing model is likely to contain more than the basic seven terms, probably several times as many, each one making a contribution to whichever of the seven is most appropriate or convenient. Consequently, the TCS must contain code that populates in real time the internal pointing model to follow the behavior of the operational model.

3.3.6 Hotspot

The hotspot, sometimes called the “pointing origin”, is the position in the focal plane that receives the image of the target. The **spkP0** structure contains two **doubles**:

- **x**: *x*-coordinate
- **y**: *y*-coordinate

These coordinates are in the rotating focal plane, so that in the case of a camera mounted on an instrument rotator a specific pixel will have a fixed $[x, y]$. The units are whatever was chosen for the focal length in the **spkOPT** structure.

See Section 6 for information about the sign conventions.

3.3.7 Mount and rotator angles

The pointing algorithm needs estimates of the current orientation of the three drives (roll, pitch, rotator). The **spkAX3** structure contains three **doubles**, all angles in radians:

- **a**: achieved roll (minus hour angle or 180° minus azimuth)
- **b**: achieved pitch (declination or elevation)
- **r**: achieved rotator angle

At first sight it may appear strange that an algorithm the principal use of which is to determine the orientations of the three drives should need *a priori* values for these very angles. The reasons are different for the mount and rotator respectively:

- The mount roll and pitch will be made available to the TCS application in anticipation of being needed for certain operational pointing corrections, for example gear errors. However, because such pointing corrections will not change much over small areas of sky there is no need for the supplied roll and pitch estimates to be particularly precise.
- The rotator angle is needed in order to deal with off-axis pointing origins (*i.e.* where the target's image is to be placed). Again, the precision requirements are fairly lax, depending how far the pointing origin is from the rotator axis.

Suitable values for populating the `spkAX3` structure may be obtained by extrapolating the current motions, using either encoder readings or the predicted demands as the basis.

See Section 6 for information about the sign conventions.

3.3.8 Time

The `spkUTC` structure contains a UTC date and time:

- `iy` : year CE
- `mo` : month (1-12)
- `id` : day (1-31)
- `ih` : hour (0-23)
- `mi` : minute (0-59)
- `sec`: seconds (0.0-61.0)

All the data members are `ints` except for `sec`, which is a `double`. The smallest illegal seconds field is almost always 60.0, but would be 61.0 *during* a positive leap second, which though a very uncommon case is correctly handled by the SOFA tools used by `spkAstr`.

It should be noted that in order to keep track of UTC leap seconds, SOFA includes a function `iauDat` that for a given UTC returns the $\text{TAI} - \text{UTC} = \Delta\text{AT}$ value in force at that time. Consequently, in order to retain correct handling of leap seconds in the long term, the TCS application must be relinked with the latest SOFA library from time to time, say once a year, so that any updated `iauDat` can be picked up. A preferable alternative, explicitly allowed by the SOFA usage conditions, is for there to be a local variant of `iauDat`, for example one that obtains the current ΔAT from an internet source or even returns a fixed value manually updated whenever a leap second occurs.

3.3.9 Astrometry

The `spkAST` structure contains the target-independent parameters used by the SOFA astrometric functions:

- `tai`: TAI MJD ($\text{JD} - 2400000.5$)
- `astrom`: star-independent astrometry quantities
- `eo`: equation of the origins (ERA–GST, radians)

The members `tai` and `eo` are doubles; `astrom` is a SOFA `iauASTROM` structure.

For an observed $[Az, El]$ target (see 3.3.10, below) the `spkAST` argument will not be accessed and its contents arbitrary.

3.3.10 Target

The target is the pointing direction in the sky. The `spkTAR` structure contains the following:

- `sys`: reference system, ICRS, APPT or AZEL (see below)
- `a`: right ascension or left-handed azimuth
- `b`: declination or elevation

The spherical coordinates `a` and `b` are doubles and in radians; `sys`, the reference system, is made with the enumeration `spkSYS`:

- ICRS $[\alpha, \delta]$: International Celestial Reference System. This is close (maximum difference less than 25 mas) to mean J2000.0.
- APPT $[\alpha, \delta]$: geocentric apparent place.
- AZEL $[Az, El]$: observed (*i.e.* as affected by refraction) azimuth and altitude. Azimuth is zero for due north and 90° for due east; altitude is called “elevation” in the present document. This is a left-handed system; internally SPK uses a south through east system, and this right-handed convention applies to one of the arguments returned by the `spkVtel` function.⁶

⁶The need for a TCS to support two left-handed coordinate systems, namely $[h, \delta]$ and $[Az, El]$, is an ever-present tripwire for the software developer. Constant vigilance is essential, and it is strongly recommended that in code referring to these coordinates or right-handed equivalents there is **always** a comment spelling out which variant is being used in that particular instance.

3.4 The three SPK functions

To use SPK, an application carries out the following steps:

1. Initialize or update all the required data structures (3.3).
2. Call either `spkAstr` (p26) to refresh or `spkCtar` (p28) to update the astrometry parameters.
3. Call `spkVtel` (p32) to solve for one of the three virtual-telescope end-points, *i.e.* axis demands, image position or target coordinates.

Step 2 can be omitted in the special case of an **ALTAZ** target or if the time has not changed.

3.4.1 `spkAstr`

The `spkAstr` function provides star-independent astrometric quantities (precession matrices *etc.*) for the UTC specified by the caller. For the utmost accuracy, but at significant computational cost, the function can be called specifying a full update, but it also offers the cheaper option of updating only that part that depends on Earth rotation, which most of the time will be good enough. For further details see pp 26-27.

3.4.2 `spkCtar`

The `spkCtar` function accepts a target specified as an epoch J2000.0 ICRS catalog $[\alpha, \delta]$ and calculates its current ICRS astrometric place ready to be used with SPK. (The astrometric place changes so slowly that in most TCS applications it can be used for the entire observing session.) For further details see p28.

3.4.3 `spkVtel`

The `spkVtel` function implements the mount pointing formulas set out in paper W02. Its power comes from the fact that it supports all three pathways through Figure 1 and so can provide many different capabilities. The call accepts (i) an option code to specify which of the three pathways is to be taken, together with (ii) a collection of data structures that define the context for the calculation; it then returns an array of values that depends on the option code `isoln` as follows:

- For the **AXES** option the values returned are the mount $[roll, pitch]$ and the position angle of the rotating focal plane’s y -axis. There may be two sets of $[roll, pitch]$, the second corresponding to a “beyond the pole” posture.
- For the **TARG** option, a celestial position is returned.

- For the **HOTS** option, the image $[x, y]$ is returned. typically one of the $[\alpha, \delta]$ s.

In addition (and even if no option is specified), the pointing direction is returned in right-handed $[Az, El]$ and $[roll, pitch]$ for use when calculating terms in the operational pointing model.

The principal application for the **AXES** option is to point and track the telescope.

The **TARG** option might be used to establish the celestial world coordinate system of the focal plane. Another application would be measuring the $[\alpha, \delta]$ of a guide star that has just been acquired so that it can be used to detect future drifting of the image.

The **HOTS** option might be used to predict the $[x, y]$ of a guide star image of known $[\alpha, \delta]$; the guider would compare this to the measured $[x, y]$ and take guiding action (such as changing the **ga** and **gb** members of a pointing model data structure).

Full details of **spkVtel** are given on pp [32-35](#).

3.5 Management of time

SPK has no direct connection with the system clock or any other timing signals, leaving these aspects completely to the TCS application. The pointing calculations are with respect to whatever UTC the caller provides when the **spkAstr** function is called, and indeed there is nothing to prevent different UTCs being used in different contexts, all proceeding in parallel. An example of this might be to explore the future track so as to establish by trial and error when cable-wrap or horizon limits will be reached.

A UTC time and date is supplied to **spkAstr** as an instance of an **spkUTC** data structure, containing the fields year, month, day, hour, minute, seconds, together with an offset in seconds. The user may elect to supply a fresh **spkUTC** each time, with the offset always left at zero, or alternatively can leave the **spkUTC** fixed and specify different times by using the offset alone. The latter choice has the advantage that the full date and time has to be formatted just once, when the system starts, and can then run on for the rest of the session purely by using the offset.

4 The demonstration programs

SPK comes with two demonstration programs, EQUAT and ALTAZ, that for the respective mount types show how the three transformations offered by the `spkVtel` function can be used by a TCS to implement not only pointing and tracking but also such things as guiding and the mapping between celestial and focal plane coordinates.

The two programs are very similar, differing only in which sort of mount is declared during initialization, the names given to the mount angles in reports, and the labels for pointing-model terms. This similarity reflects how SPK, and the “virtual telescope” principle more generally, is based on there being little or no difference between how the two types of mount are controlled. **This important principle may be overlooked by the TCS developer.** For example, in order to displace a star image vertically in the field of view of an altazimuth telescope it may seem obvious that all that needs to be done is to introduce an offset into the demands going to the elevation drive. *But this is not the right thing to do.* The offset should instead be applied in terms of focal plane coordinates for a target being tracked in $[Az, El]$, just as it would be in the equatorial case. Doing so will deal with small field orientation effects caused by refraction and the pointing model, as well as differential refraction. The motion, almost but not entirely a change in the demands going to the elevation drive, will be a *consequence* of the changed pointing prescription, not something the developer should be conscious of.

Because the EQUAT and ALTAZ programs are almost identical, the description below applied to both, with the occasional differences mentioned. Certain portions of the code are omitted for brevity; in particular, reports are reproduced but not the code that writes them.

4.1 Preliminaries

The code begins as a *pro forma* `main()` program and then declares some useful constants. Instances (in a couple of cases more than one) of the various data structures are then declared; these will become arguments of the SPK function calls, `spkVtel` in particular:

```

spkEOP eop;      /* polar motion and UT1-UTC (from IERS) */
spkOBS obs;      /* site location and mount type */
spkAIR air;      /* local weather readings */
spkOPT opt;      /* focal length and color */
spkPM pm;        /* 7-term pointing model */
spkAX3 ax3;      /* achieved roll, pitch, rotator angles */
spkPO po;        /* image position in focal plane */
spkUTC utc;      /* UTC date and time */
spkAST ast;      /* target-independent astrometric parameters */
spkTAR tar;      /* sky target */

spkTAR tar2;     /* another target */
spkPO po2;       /* another hotspot */

```

In order to make the reports clearer, names for the different supported target reference systems are declared:

```
static char *csys[] = {
    "illegal",
    "ICRS",
    "apparent",
    "altaz"
};
```

Miscellaneous variables are declared, with one small difference between the `altaz.c` and `equat.c` cases for cosmetic reasons.

The executable code begins with initialization of the data structures, the first being the Earth orientation parameters:

```
eop.xp = 50.995e-3 * AS2R; /* polar motion x */
eop.y = 376.723e-3 * AS2R; /* polar motion y */
eop.dut1 = 155.0675e-3; /* UT1-UTC */
```

The EOPs are available from the International Earth Rotation and reference systems Service. An advanced TCS might obtain the parameters automatically via the internet; a simpler method is to read an initialization file that the observatory updates every few days. For a TCS with modest accuracy objectives the EOPs could simply be set to zero, though in the case of UT1–UTC this can lead to pointing errors of nearly 15 arcseconds.⁷

Next, details of the observatory are set up, namely geographical position and type of telescope mount; there is also an opportunity to declare a “no go” area around the mount pole, to prevent unrealistically rapid motion and other unruly behavior:

```
iauAf2a ( '-', 5, 41, 54.2, &obs.slons ); /* east longitude */
iauAf2a ( '-', 15, 57, 42.8, &obs.slat ); /* geodetic latitude */
obs.sh = 625.0; /* altitude */
obs.mount = ALTAZ; /* mount type */
obs.pavoid = 0.5 * D2R; /* zenith avoidance */
```

... and in the equatorial case:

⁷The situation could get worse if proposals to cease UTC leap seconds, first made in the early 2000s, are eventually put into effect. The safest thing is for all TCSs to take into account UT1–UTC whatever the accuracy aspirations.

```

:
obs.mount = EQUAT;           /* mount type */
obs.pavoid = 0.5 * D2R;      /* pole avoidance */
:

```

Local air conditions, needed for calculating atmospheric refraction, are declared:

```

air.p = 952.0;  /* pressure (hPA=mB) */
air.t = 18.5;   /* temperature (Celsius) */
air.h = 0.83;   /* humidity */

```

How frequently this information is updated depends on the TCS and the accuracy objectives, but once per new target may be adequate (or in some cases even once per observing session). Continuous updating from an electronic weather station is the ultimate solution, but some form of smoothing is essential if noisy readings are not to inject jitter into the telescope track. (See also the earlier note about sea-level pressure QNH versus local pressure QFE; it is the latter that SPK needs.)

The information SPK needs about the telescope optical configuration is the focal length and the observing wavelength:

```

opt.fl = 3000.0;  /* focal length (mm) */
opt.wl = 0.55;    /* wavelength (micrometers) */

```

The only significance of focal length in the pointing calculation is that it defines the units in which $[x, y]$ image positions will be expressed. Entering the focal length as 1.0 would not affect the pointing calculations except that focal-plane $[x, y]$ coordinates are essentially radians. The wavelength plays a more fundamental part in that it affects the refraction calculation.

The starting amplitudes for the seven terms of the pointing model are set, together with small “trimming” adjustments to two of them:

```

pm.pia = 25.0 * AS2R;  /* +IA */
pm.pib = 15.0 * AS2R;  /* +IE */
pm.pnp = 8.0 * AS2R;   /* +CA */
pm.pca = -110.0 * AS2R; /* +NPAE */
pm.paw = 999.9 * AS2R; /* +AW */
pm.pan = -12.0 * AS2R; /* +AN */

pm.ga = 0.0 * AS2R;    /* guiding correction to pm.pca */
pm.gb = 0.0 * AS2R;    /* guiding correction to pm.pib */

```

...with only comment changes in the equatorial case:

```

:
pm.pia = 25.0 * AS2R; /* -IH */
pm.pib = 15.0 * AS2R; /* +ID */
pm.pnp = 8.0 * AS2R; /* -CH */
pm.pca = -110.0 * AS2R; /* -NP */
pm.paw = 999.9 * AS2R; /* -MA */
pm.pan = -12.0 * AS2R; /* +ME */
:

```

During the operation of the TCS the model will be updated, both continuously to allow for the effects of telescope attitude and abruptly to achieve special effects.

Starting values for the three axis positions are set:

```

ax3.a = 0.0 * D2R; /* achieved azimuth (south zero right-handed) */
ax3.b = 0.0 * D2R; /* achieved elevation */
ax3.r = 30.0 * D2R; /* achieved rotator angle */

```

...and in the equatorial case:

```

:
ax3.a = 0.0 * D2R; /* achieved -HA (right-handed) */
ax3.b = 0.0 * D2R; /* achieved Dec */
:

```

These three values will be actively controlled as tracking proceeds, from a combination of encoder readings and extrapolation.

A pointing origin (hotspot) is declared:

```

po.x = 10.0;
po.y = 20.0;

```

More typically the initialization process would set the pointing origin to $[0, 0]$ (*i.e.* the optical axis) and leave it up to the TCS application to manage where in the focal plane the image is being aimed at any given moment.

The starting UTC is specified:

```

utc.iy = 2013;    /* year (CE) */
utc.mo = 4;      /* month (1-12) */
utc.id = 2;      /* day (1-28,29,30,31) */
utc.ih = 23;     /* hour */
utc.mi = 15;     /* minute */
utc.sec = 43.55; /* seconds (0-59.999... or 0-60.999...) */

```

In a real TCS this would of course come from a clock.

The star-independent astrometry parameters for that moment can now be calculated:

```

if ( spkAstr ( 1, utc, 0.0,
              &eop, &obs, &air, &opt, &ast ) ) return -1;

```

A celestial target is specified, starting from a catalog entry:

```

if ( iauTf2a ( ' ', 14, 34, 16.81183, &rc ) ) return -1;
if ( iauAf2a ( '-', 12, 31, 10.3965, &dc ) ) return -1;
pr = atan2 ( -354.45e-3 * AS2R, cos(dc) );
pd = 595.35e-3 * AS2R;
px = 164.99e-3;
rv = 0.0;
spkCtar ( rc, dc, pr, pd, px, rv, &ast, &tar );

```

We now specify the image orientation, the position angle with respect to the ICRS meridian of $+y$ at the specified hotspot:

```

ypa = 30.0 * D2R;

```

The choice of 30° is merely illustrative; it would be much more typical to use zero and an on-axis hotspot, meaning the focal plane's y -axis will point north.

4.2 Tracking

We are now ready to illustrate sidereal tracking. This is the part of the `EQUAT` and `ALTAZ` demonstrations that least resembles what an actual TCS would do, and is only a general indication of what calculations might be involved. The code shows just one way of computing the tracking demands, and gives some indication of the level of intricacy that a real TCS application would require.

At first sight all that is involved is to take the current target and hotspot and solve for the axis demands. However, there is an interplay between the three axes that has to be resolved, and the servo systems controlling the axis may well expect velocity feedforward information (and conceivably acceleration and higher derivatives). Consequently generating each new set of demands is likely to involve multiple VT solutions, and the precise way this is done will be something for the application developer to decide.

The demonstration starts one second before the UTC that has been specified, and at intervals of 0.05 SI seconds computes the demand angles for the three axes. We will also estimate velocities in the three axes; this is to enable extrapolation to the next iteration but might also be needed for servo reasons. It should also be noted that servo engineers tend to disapprove of even tiny irregularities in the tracking demands, and arranging the computations to avoid, or at least minimize, glitches in the demand stream is highly desirable.

We are assuming that the astrometry parameters have been computed for a particular moment using `spkAstr` with `jfull = TRUE`, and that the track is relative to that time and involves only `jfull = FALSE` calls to `spkAstr`. Some TCS developers will decide to call `spkAstr` with `jfull = TRUE` every tracking step and accept the (considerable) computational expense. Most will opt instead for occasional (say once a minute) full updates, maintaining precision long-term at the expense of (sub-milliarcsecond) glitches. A full refresh only at the start of a new track is a workable plan, incurring errors of less than $0''.1$ for tracks several hours long and avoiding glitches (from this cause at least) completely.

In the demonstration code, updating the pointing model terms is performed each time through, but an economical implementation could elect to do it less frequently, say at 1 Hz. If the update intervals are too long, the corrections will not keep up with changing flexure *etc.* and will also inject small glitches into the tracking demands.

A noteworthy feature of the demonstration is that the velocities are **not** computed by differencing successive demands, though this approach would be efficient and may well look much as it does in the present simulation. The potential difficulty is in the general case, where the observing program is free to introduce abrupt changes to target, hotspot *etc.* The resulting transients can trigger adverse reactions in servo systems, so the demonstration code instead recomputes the demands from last time, but using the latest target and hotspot. Though computationally wasteful, this simple measure means that the velocities always reflect a steady background track.

The first step in the demonstration is to initialize various quantities and then enter the 1 second-at-20-Hz loop:

```

/* Initialize the per-timestep changes in the three axes. */
da = 0.0;
db = 0.0;
dr = 0.0;

/* Initialize the time offset. */
dt = 0.0;

/* The demonstration track (20Hz for 1s). */
for ( i = 0; i <= 20; i++ ) {

```

Each tracking update starts by solving for the demands *last time*, but using the current circumstances in case there have been abrupt changes:

```

/* Solve for previous roll,pitch demands and field orientation. */
j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
              &tara, &tare, &tarr, &tarp, soln );
:

/* The roll, pitch and rotator demands then. */
aold = soln[0];
bold = soln[1];
rold = iauAnpm ( soln[4] + ypa );

```

We now turn our attention to the present UTC, expressing it as an offset to the time originally supplied, and we update the astrometry to take account of Earth rotation:

```

/* New time offset (seconds) relative to provided UTC. */
dt = ( (double) (i-20) ) / 20.0;

/* Update astrometry to that time. */
if ( spkAstr ( 0, iy, mo, id, ih, mi, sec, dt, &eop, &obs,
              &air, &opt, &ast ) ) return -1;

```

We extrapolate the axis angles and recalculate, twice to let things settle, and noting the changes since last time:

```

/* Iterate to let roll/pitch/rotator angles settle. */
for ( it = 0; it < 2; it++ ) {

    /* Extrapolated encoder readings. */
    ax3.a = aold + da;
    ax3.b = bold + db;
    ax3.r = rold + dr; iauAnpm ( soln[4] + ypa );

    /* Solve for roll,pitch demands and field orientation. */
    j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
                  &tara, &tare, &tarr, &tarp, soln );
    :

    /* Set achieved roll,pitch,rotator to match. */
    ax3.a = soln[0];
    ax3.b = soln[1];
    ax3.r = iauAnpm ( soln[4] + ypa );

    /* Changes since last time. */
    da = iauAnpm ( ax3.a - aold );
    db = iauAnpm ( ax3.b - bold );
    dr = iauAnpm ( ax3.r - rold );

    /* Next iteration. */
}

```

Next we update the pointing model, which may have changed slightly as a result of the change in attitude. The altazimuth...

```

pm.pia = 25.0 * AS2R;          /* +IA */
pm.pib = 15.0 * AS2R;          /* +IE */
pm.pnp = 8.0 * AS2R;           /* +CA */
pm.pca = -110.0 * AS2R;        /* +NPAE */
pm.paw = 999.9 * AS2R;         /* +AW */
pm.pan = -12.0 * AS2R;         /* +AN */
pm.pvd = 10.0 * AS2R * cos(tare); /* +TF */

```

... and equatorial...

```

pm.pia = 25.0 * AS2R;          /* -IH */
pm.pib = 15.0 * AS2R;          /* +ID */
pm.pnp = 8.0 * AS2R;           /* -CH */
pm.pca = -110.0 * AS2R;        /* -NP */
pm.paw = 999.9 * AS2R;         /* -MA */
pm.pan = -12.0 * AS2R;         /* +ME */
pm.pvd = 10.0 * AS2R * cos(tare); /* +TF */
:
```

... cases are the same apart from the comments.

The tracking illustration is complete. In the altazimuth case the report is as follows:

TAI	azimuth	elevation	rotator	
56384.9696579861	91.5962372761	47.3611395498	309.8100307422	deg
	-3.6793457591	+14.4476380752	-0.6156716180	"/s
56384.9696585648	91.5961861739	47.3613402043	309.8100222021	deg
	-3.6793542193	+14.4476377656	-0.6148119632	"/s
56384.9696591435	91.5961350716	47.3615408588	309.8100136492	deg
	-3.6793626300	+14.4476375408	-0.6152177327	"/s
MJD				

The equatorial report shows an HA tracking rate of about 15 arcseconds per sidereal second and approximately zero in elevation and rotator angle:

TAI	hour angle	declination	rotator	
56384.9696579861	-43.6619835316	-13.0082797783	210.2268042522	deg
	+15.0454749917	+0.0546843530	+0.0555711106	"/s
56384.9696585648	-43.6617745739	-13.0082790175	210.2268050235	deg
	+15.0454783351	+0.0546895851	+0.0556434599	"/s
56384.9696591435	-43.6615656161	-13.0082782567	210.2268057997	deg
	+15.0454790803	+0.0546907798	+0.0557620306	"/s
MJD				

4.3 The three pathways

We are ready to demonstrate the three pathways through Figure 1 that `spkVtel` implements. These are the basic ingredients that can be combined to achieve not only the key functions of pointing, tracking and autoguiding, but also establishing an image's world coordinate system, offsetting between acquisition camera and instrument or between multiple feed horns, trailing along a vertical spectrograph slit, chopping between source and sky, and so on.

4.3.1 Target and hotspot to mount angles

The first pathway is just a repeat of the final call of the tracking demonstration; we know the target $[\alpha, \delta]$ and hotspot $[x, y]$, and wish to calculate the corresponding mount $[roll, pitch]$ and also the field orientation, requiring `spkVtel`'s `isoln` option to be AXES:

```
j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
              &tara, &tare, &tarr, &tarp, soln );
```

The report from `altaz.c` is as follows:

```
Given:
  TAI = 56384.969659144 MJD
  RMA = -50.189986351 deg
  target = 14 34 16.4960 -12 31 02.524 ICRS
  hotspot [x,y] = 10.000000, 20.000000 mm
Returned:
  encoder [Az,El] = 91.596135072 (N thru E), 47.361540852 deg
  PA of +y = 29.999999998 deg (ICRS)
```

The quoted position angle is the direction of $+y$ at the pointing origin, with respect to the stated celestial coordinate system, in this case ICRS. In other words, zero would mean that at the pointing origin $+y$ points north. If exactly the same pointing direction in the sky were to be specified as a geocentric apparent place, the position angle would again be different, by up to 180° for a point between the ICRS and apparent poles. The `equat.c` report is similar, differing only in the mount axis angles:

```
Given:
  TAI = 56384.969659144 MJD
  RMA = 210.226805800 deg
  target = 14 34 16.4960 -12 31 02.524 ICRS
  hotspot [x,y] = 10.000000, 20.000000 mm
Returned:
  encoder [HA,Dec] = 43.661565623 (W +ve), -13.008278256 deg
  PA of +y = 30.000000008 deg (ICRS)
```

4.3.2 Mount angles and hotspot to target

The second pathway through Figure 1 is where everything stays the same except that the pointing origin $[x, y]$ has changed to $[25.0, 27.5]$ and we set `isoln = TARG`:

```
po2.x = 25.0;
po2.y = 27.5;
j = spkVtel ( TARG, &obs, &opt, &pm, &ast, &ax3, &tar, &po2,
              &tara, &tare, &tarr, &tarp, soln );
```

The reports show that for the same mount and rotator demands the new hotspot corresponds to a new $[\alpha, \delta]$. In the `altaz.c` case:

```
Given:
  TAI = 56384.969659144 MJD
  RMA = 309.810013649 deg
  hotspot [x,y] = 25.000000, 27.500000 mm
  encoder [Az,El] = 91.596135072 (N thru E), 47.361540859 deg
Returned:
  target = 14 35 35.1364 -12 32 10.772 ICRS
```

...and for `equat::`

```
Given:
  TAI = 56384.969659144 MJD
  RMA = 210.226805800 deg
  hotspot [x,y] = 25.000000, 27.500000 mm
  encoder [HA,Dec] = 43.661565616 (W +ve), -13.008278257 deg
Returned:
  target = 14 35 35.1364 -12 32 10.772 ICRS
```

4.3.3 Mount angles and target to hotspot

The third and final pathway through Figure 1 accepts a target and for given mount and rotator demands predicts the focal-plane $[x, y]$ that will receive the image. For the demonstration the mount and rotator demands are left as they are and we use as a target the $[\alpha, \delta]$ found in the previous example, with `spkVtel`'s `isoln` set to `HOTS`:

```

if ( iauTf2a ( ' ', 14, 35, 35.1363560, &rc ) ) return -1;
if ( iauAf2a ( '-', 12, 32, 10.771790, &dc ) ) return -1;
spkCtar ( rc, dc, 0.0, 0.0, 0.0, 0.0, &ast, &tar2 );
j = spkVtel ( HOTS, &obs, &opt, &pm, &ast, &ax3, &tar2, &po,
              &tara, &tare, &tarr, &tarp, soln );

```

The code is identical in `altaz.c` and `equat.c`. The `altaz.c` report is:

```

Given:
  TAI = 56384.969659144 MJD
  RMA = 309.810013649 deg
  target = 14 35 35.1364 -12 32 10.772 ICRS
  encoder [Az,El] = 91.596135072 (N thru E), 47.361540859 deg
Returned:
  hotspot [x,y] = 25.000000, 27.500000 mm

```

...recovering the test $[x, y]$ that we used. The `equat.c` report is similar:

```

Given:
  TAI = 56384.969659144 MJD
  RMA = 210.226805800 deg
  target = 14 35 35.1364 -12 32 10.772 ICRS
  encoder [HA,Dec] = 43.661565616 (W +ve), -13.008278257 deg
Returned:
  hotspot [x,y] = 25.000000, 27.500000 mm

```

That completes the three Figure 1 pathways.

4.4 Direction of the vertical

Even though a target is usually tracked in $[\alpha, \delta]$, and the field orientation requirement is given relative to north, it is often necessary to know which direction in the focal plane corresponds to the vertical. This is easy to do, simply by specifying the pointing direction as an $[Az, El]$ target but leaving other `spkVtel` arguments as they were. The latest call to `spkVtel` has already returned the pointing direction as an $[Az, El]$, so the new target is simply that but in left-handed form:

```

tar2.sys = AZEL;
tar2.a = D180 - tara;
tar2.b = tare;
j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar2, &po,
              &tara, &tare, &tarr, &tarp, soln );

```

Note that the original hotspot has been retained, so it is the vertical through that point that we are interested in. The `altaz.c` report is:

```

Given:
  TAI = 56384.969659144 MJD
  RMA = 309.810013649 deg
  target = 88.499802127 (N thru E), 46.938342447 deg (altaz)
  hotspot [x,y] = 10.000000, 20.000000 mm
Returned:
  encoder [Az,El] = 91.596135072 (N thru E), 47.361540859 deg
  PA of +y = 129.948849391 deg (altaz)

```

... and the `equat.c` report is similar:

```

Given:
  TAI = 56384.969659144 MJD
  RMA = 210.226805800 deg
  target = 88.499802127 (N thru E), 46.938342447 deg (altaz)
  hotspot [x,y] = 10.000000, 20.000000 mm
Returned:
  encoder [HA,Dec] = 43.597998256 (W +ve), -11.016168751 deg
  PA of +y = 129.948849393 deg (altaz)

```

The EQUAT and ALTAZ demonstrations end at this point.

4.5 The POINT testbed

The EQUAT and ALTAZ programs just described demonstrate all three virtual-telescope transformations supported by the `spkVtel` function. Though the techniques used are perfectly straightforward, and quite representative of how a real TCS would be coded, some potential SPK users may prefer to start with something even more primitive. To satisfy this need, SPK comes with a testbed program called POINT (Section 4.5) that performs the basic pointing/tracking calculation for a hardwired test case. It takes a set of circumstances (site

location, star coordinates, time *etc.*) and calculates the encoder demands that the TCS would make in order to acquire the target. This is done in the simplest possible way, with no regard for computational efficiency. A particular use for the `point.c` code is as a template for experimentation, and a source of reference data for sanity checks and other diagnostic purposes.

POINT is merely a front-end for a function (`spkS2e`, described on pp 29-31, that accepts the 40 parameters needed to specify the test case and generates the corresponding encoder demands. It is worth noting that although `spkS2e` could itself be used in a TCS, it is computationally very wasteful compared with the methods shown in EQUAT and ALTAZ. However, its simplicity – a single function call does the entire pointing/tracking calculation – may make it attractive for use in a minimal TCS, and in fact it can run at kilohertz rates on any modern processor.

5 Function specifications

spkAstr	<i>update astrometry context</i>	spkAstr
----------------	----------------------------------	----------------

CALL :

```
j = spkAstr ( jfull, utc, dt, &eop, &obs, &air, &opt, &ast );
```

ACTION :

For a specified UTC, update the star-independent astrometry parameters.

GIVEN :

<i>jfull</i>	int	TRUE/FALSE = full/partial update (Notes 1,2)
<i>utc</i>	spkUTC	UTC date and time (Notes 2-4)
<i>dt</i>	double	offset (SI seconds, Note 2)
<i>eop</i>	spkEOP*	Earth orientation parameters
<i>obs</i>	spkOBS*	site location and mount type
<i>air</i>	spkAIR*	ambient weather conditions
<i>opt</i>	spkOPT*	for color

RETURNED :

<i>ast</i>	spkAST*	astrometry parameters (Note 1)
------------	---------	--------------------------------

RETURNED (function value) :

int	status: +3 = both of next two
	+2 = time is after end of day (Note 3)
	+1 = dubious year (Note 4)
	0 = OK
	-1 = bad year
	-2 = bad month
	-3 = bad day
	-4 = bad hour
	-5 = bad minute
	-6 = bad second (< 0)
	-7 = other error (Note 5)

NOTES :

1. The flag `jfull` selects whether the entire set of star-independent astrometry parameters are refreshed (`jfull = TRUE`) or just the Earth rotation angle. Other quantities such as precession matrices are not recomputed in the `jfull = FALSE` case, and will as time goes on become less accurate. How frequently the parameters are fully refreshed is a decision for the application developer, but once for each new target could be adequate. The TAI included in the star-independent astrometry parameters corresponds to the ERA.

2. The UTC date and time is supplied as an `spkUTC` structure, containing `y,m,d,h,m,s` fields, with an offset in seconds. The user may elect to supply a fresh `spkUTC` structure each time, with the offset always zero, or can supply the same `spkUTC` structure each time and move forward using the offset alone.
3. The warning status “time is after end of day” usually means the second argument is greater than 60.0. However, in a day ending in a leap second the limit changes to 61.0 (or 59.0 in the case of a negative leap second).
4. The warning status “dubious year” flags UTCs that predate the introduction of the time scale or that are too far in the future to be trusted. See the `iauDat` function for further details.
5. The error status `-7` should be impossible.

spkCtar	<i>accept ICRS catalog star as target</i>	spkCtar
----------------	---	----------------

CALL :

```
spkCtar ( rc, dc, pr, pd, px, rv, &ast, &tar );
```

ACTION :

Transform an epoch J2000.0 ICRS star catalog entry into an ICRS astrometric target.

GIVEN :

<i>rc</i>	double	ICRS right ascension at J2000.0 (radians, Note 1)
<i>dc</i>	double	ICRS declination at J2000.0 (radians, Note 1)
<i>pr</i>	double	α proper motion (radians/year, Note 2)
<i>pd</i>	double	δ proper motion (radians/year)
<i>px</i>	double	parallax (arcsec, Note 3)
<i>rv</i>	double	radial velocity (km/s, +ve if receding)
<i>ast</i>	spkAST*	astrometry parameters

RETURNED :

<i>tar</i>	spkTAR*	target, ICRS astrometric
------------	---------	--------------------------

NOTES :

1. Star data for an epoch other than J2000.0 (for example from the Hipparcos catalog, which has an epoch of J1991.25) will require preliminary treatment.
2. The proper motion in α is $\dot{\alpha}$ rather than $\dot{\alpha} \cos \delta$.
3. In catalogs where the α and δ proper motions are given in milliarcseconds per year, both will need scaling into radians before the μ_α value is divided by $\cos \delta$. *n.b.* Some catalogs quote proper motions per century rather than per year.
4. The parallax value is in arcseconds rather than radians as it is a distance measure rather than an angle.

spkS2e*One-call star to mount calculation***spkS2e**

CALL :

```
j = spkS2e ( xp, yp, dut1, slon, slat, sh, mount, pnogo, p, tc, h, fl, wl,
             pia, pib, pvd, pca, pnp, paw, pan, ga, gb, x, y, a, b,
             r, iy, im, id, ih, mi, sec, rc, dc, pr, pd, px, rv,
             &adem, &bdem, &ademb, &bdemb, &rdem );
```

ACTION :

The function performs the end-to-end pointing transformation, starting with an ICRS star catalog entry and predicting the mount encoder demands required to acquire the star image. It does so in the simplest possible way, by first populating the SPK data structures and then calling **spkAstr**, **spkCtar** and **spkVtel**, with no consideration for computational efficiency.

Its main purpose is as the core component of the SPK demonstrator **POINT**, though it could be used in a basic telescope control system where only the pointing/tracking capability of SPK is needed and efficiency is not an important consideration.

GIVEN :

Earth orientation parameters (Note 1)

<i>xp</i>	double	polar motion <i>x</i> coordinate (radians)
<i>yp</i>	double	polar motion <i>y</i> coordinate (radians)
<i>dut1</i>	double	Δ UT1: UT1–UTC (seconds)

Observatory

<i>slon</i>	double	site east longitude (ITRF, radians)
<i>slat</i>	double	site geodetic latitude (ITRF, radians)
<i>sh</i>	double	site altitude above WGS84 ellipsoid (m)
<i>mount</i>	int	mount type: 1 = equatorial, else = altazimuth
<i>pnogo</i>	double	closest to mount pole allowed (radians)

Ambient air conditions

<i>p</i>	double	pressure (hPa \equiv mB)
<i>tc</i>	double	temperature (degrees Celsius)
<i>h</i>	double	relative humidity (0.0–1.0)

Optics

<i>fl</i>	double	focal length (user units)
<i>wl</i>	double	wavelength (μ m)

Pointing model coefficients (radians)

<i>pia</i>	double	IA: roll ($-\text{HA}$ or $\pi-\text{azimuth}$) index error
<i>pib</i>	double	IB: pitch (declination or elevation) index error
<i>pvd</i>	double	VD: droop
<i>pca</i>	double	CA: telescope/pitch nonperpendicularity
<i>pnP</i>	double	NP: roll/pitch nonperpendicularity
<i>paw</i>	double	AW: roll axis misalignment across meridian
<i>pan</i>	double	AN: roll axis misalignment along meridian
<i>ga</i>	double	ΔCA guiding correction
<i>gb</i>	double	ΔIB guiding correction

Hotspot

<i>x</i>	double	<i>x</i> -coordinate in rotating focal plane (user units)
<i>y</i>	double	<i>y</i> -coordinate in rotating focal plane (user units)

Mount and rotator angles (achieved)

<i>a</i>	double	achieved $-\text{HA}$ or $\pi-\text{Az}$ (radians)
<i>b</i>	double	achieved Dec or El (radians)
<i>r</i>	double	achieved rotator angle (radians, Note 3)

UTC

<i>iy</i>	int	year CE
<i>im</i>	int	month
<i>id</i>	int	day
<i>ih</i>	int	hour
<i>mi</i>	int	minute
<i>sec</i>	double	seconds

Target

<i>rc</i>	double	ICRS α at epoch J2000.0 (radians, Note 2)
<i>dc</i>	double	ICRS δ at epoch J2000.0 (radians, Note 2)
<i>pr</i>	double	μ_α (radians/year, Note 2)
<i>pd</i>	double	μ_δ (radians/year)
<i>px</i>	double	parallax (arcsec, Note 3)
<i>rv</i>	double	radial velocity (km/s, +ve if receding)

RETURNED :

<i>adem</i>	double*	demand $-\text{HA}$ or $\pi-\text{Az}$ (radians)
<i>bdem</i>	double*	demand Dec or El (radians)
<i>ademb</i>	double*	demand $-\text{HA}$ or $\pi-\text{Az}$ (radians, Note 4)
<i>bdemb</i>	double*	demand Dec or El (radians, Note 4)
<i>rdem</i>	double*	demand rotator angle (radians, Note 5)

NOTES :

1. The Earth Orientation Parameters can be obtained from tabulations provided by the International Earth Rotation and Reference Systems Service.

x_p and **y_p** are the coordinates (in radians) of the Celestial Intermediate Pole with respect to the International Terrestrial Reference System (see IERS Conventions), measured along the meridians 0 and 90° west respectively.

The ΔUT1 changes very slowly except for UTC leap seconds, where it jumps by one second.

2. Star data for an epoch other than J2000.0 (for example from the Hipparcos catalog, which has an epoch of J1991.25) will require preliminary treatment.

The target's proper motion in α is $d\alpha/dt$ rather than $d\alpha \cos \delta/dt$. In catalogs where the α and δ proper motions are given in milliarcseconds per year, both will need scaling into radians before the μ_α value is divided by $\cos \delta$. *n.b.* Some catalogs quote proper motions per century, others per year.

The parallax value is in arcseconds rather than radians as it is a distance measure rather than an angle.

3. The achieved rotator angle **r** is zero when the $+y$ axis points toward the projection of the mount's south pole, and increases counterclockwise.
4. The second set of mount demands, **ademb**, **bdemb**, is for the "beyond the pole" solution.
5. The demand rotator angle **rdem** is that required to align the $+y$ axis to ICRS north when the mount is in its normal (*i.e.* not beyond the pole) attitude.
6. This function does not itself validate the given arguments, but does return the error status if any of the called functions fails.
7. Although grossly inefficient, it will run at kilohertz rates on any modern processor.

spkVtel	<i>solve virtual telescope</i>	spkVtel
----------------	--------------------------------	----------------

CALL :

```
j = spkVtel ( isoln, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
              &tara, &tare, &tarr, &tarp, soln[5] );
```

ACTION :

Solve a “virtual telescope” (Note 1).

GIVEN :

<i>isoln</i>	spkVTS	what to solve for (Notes 3,4)
<i>obs</i>	spkOBS*	observatory
<i>opt</i>	spkOPT*	optics (for focal length)
<i>pm</i>	spkPM*	pointing model (Note 5)
<i>ast</i>	spkAST*	astrometry parameters (Note 6)
<i>ax3</i>	spkAX3*	achieved roll, pitch, rotator angles (radians)
<i>tar</i>	spkTAR*	target (ICRS, apparent or altazimuth)
<i>po</i>	spkPO*	hotspot [<i>x, y</i>] (Note 2, same units as focal length)

RETURNED :

<i>tara</i>	double*	target observed azimuth (N thru E, radians)
<i>tare</i>	double*	target observed elevation (radians)
<i>tarr</i>	double*	target roll (radians)
<i>tarp</i>	double*	target pitch (radians)
<i>soln</i>	double[5]	solution (Notes 4,9)

RETURNED (function value) :

int	status: +1 = pole avoidance occurred
	0 = OK
	-1 = no solutions
	-2 = geometrically impossible
	-3 = illegal target system

NOTES :

1. The virtual telescope (see reference) is a data construct that describes the relationship between three aspects of pointing a telescope, namely:
 - the coordinates of the celestial target,
 - the position of the target’s image in the focal plane, and
 - the angles to which the mount axes must be set in order to acquire the image.

Given any two of these, along with certain other information, including astrometric parameters some of which are changing with time, the third can be deduced.

Thus a telescope can track a star by knowing the $[\alpha, \delta]$ and the desired image $[x, y]$, and repeatedly solving for mount angles as the astronomical transformations progress. Or the mapping between sky coordinates and focal plane coordinates can be sampled, in both directions.

2. Focal plane $[x, y]$ is left-handed when projected onto the sky. Zero rotator angle is when the projection on the sky of the y -axis points at the negative mount pole, and the angle increases counter-clockwise on the sky.
3. The argument `isoln` specifies what quantities are to be solved for:
 - If `isoln` = `AXES`, the result is the mount axis angles demands required to image the target onto the desired $[x, y]$, known as the “hotspot”. There can be two sets of these angles, one corresponding to the “beyond the pole” state of the mount. Also computed is the field orientation, in the form of the position angle of the rotating focal plane’s y -axis. *n.b.* The given argument `ax3` is needed for other purposes and must contain valid “achieved” angles.
 - If `isoln` = `HOTS`, the result is the $[x, y]$ position of the image in the rotating focal plane. The given argument `po` is not used and need not be initialized.
 - If `isoln` = `TARG`, the result is the celestial coordinates of the target. The supported systems are ICRS astrometric $[\alpha, \delta]$, geocentric apparent $[\alpha, \delta]$ and observed $[Az, El]$, the latter left-handed (north through east). The given argument `tar` is needed for other purposes and must contain a valid target.
 - If `isoln` = `DEFT` (or any other value) no solution takes place – see the next note.
4. Irrespective of the `isoln` value, the target’s observed place as right-handed $[Az, El]$ and mount $[roll, pitch]$ are returned, for general purposes including the calculation of pointing-model terms.
5. The pointing model structure `pm` contains seven coefficients, called IA, IB, VD, CA, NP, AW and AN. A typical operational pointing model would contain a greater variety, each being mapped onto the most natural one of the seven.
6. The astrometry parameters `ast` can be formed by calling the `spkAstr` function with the appropriate UTC specified. In `spkAstr` the argument `jfull` controls whether a complete recalculation is to be carried out or merely the Earth rotation angle. A common strategy would be to recompute the parameters prior to acquiring a new target (or even once a night), and to update only the ERA at the rate of the tracking loop, typically 20 Hz.
7. The returned `tara` and `tare` are the observed place (*i.e.* the pointing direction) as left-handed $[Az, El]$, where azimuth runs north through east.
8. The returned `tarr` and `tarp` are the target observed place in mount roll,pitch, called AIM in the code. For an equatorial mount, the nominal orientation of the $[roll, pitch]$ frame makes `tarr` minus hour angle and `tarp` declination. For an altazimuth mount, the nominal orientation makes `tarr` π minus the north-through-east azimuth and `tarp` elevation (more properly called altitude). For

both types of mount, the actual orientation depends on data members `pan` and `paw` in the `spkPM` pointing-model structure. These are typically a few (or a few tens of) arcseconds, and are called polar misalignment for an equatorial and azimuth axis tilt in an altazimuth.

9. The `soln` array receives from one to five doubles, depending on the `isoln` argument, as follows:

<code>isoln</code>	<code>AXES</code>	<code>HOTS</code>	<code>TARG</code>	<i>else</i>
<code>soln[0]</code>	<i>r</i>	<i>x</i>	<i>f</i>	-
<code>soln[1]</code>	<i>p</i>	<i>y</i>	<i>g</i>	-
<code>soln[2]</code>	<i>rb</i>	-	-	-
<code>soln[3]</code>	<i>pb</i>	-	-	-
<code>soln[4]</code>	θ	-	-	-

Unused `soln` elements (marked -) are undefined. Everything but *x* and *y* are radians.

For the `AXES` solution:

- The results *r, p* and *rb, pb* are the roll, pitch mount pointing demands needed to image the target onto the hotspot.
- The “*b*” pair are for the “beyond the pole” configuration. It is up to the telescope control application to decide whether *r, p* or *rb, pb* are to be used as drive demands. For equatorials the “*b*” set represents below the pole in a fork or horseshoe mount and one of the “meridian flip” cases in GEMs and cross-axis mounts. The “*b*” set are not commonly used for altazimuths, though they could be in designs where “through the zenith” is allowed.
- The θ result is the rotator demand that will, at the hotspot, make the *y*-axis of the rotating focal plane point north (or up) in the target’s celestial coordinate system.
- If the desired field orientation (*i.e.* the desired position angle of the *y*-axis, counterclockwise from north) is *ypa*, the rotator demand is $\theta + ypa$. In altazimuth telescopes in particular this will provide the ever-changing tracking demands for the rotator. The same is true for equatorials, though usually the rotator is not driven continuously and the small residual field rotations neglected.
- It is important to understand that the `soln[4]` value is for the **achieved** mount orientation, supplied in the `ax3` structure, and whether this is close to *r, p* or *rb, pb* depends on the current “beyond the pole” status.

For the `HOTS` solution, the *x, y* results are the coordinates of the target’s image given the current mount pointing. They are in the units chosen for the telescope focal length in the `spkOBS` structure, for example millimeters, and are on the rotating focal plane, so will stay constant for a specific place on an instrument, such as a spectrograph slit.

For the `TARG` solution, the *f, g* results are the sky position that corresponds to the

hotspot $[x, y]$ given the current mount pointing. This is an $[\alpha, \delta]$ when the target's system is ICRS or APPT, or $[Az, El]$ (left-handed) for the AZEL case.

6 Coordinate conventions

The coordinate conventions that SPK uses are a fruitful cause of confusion, mainly for historical reasons. The present section spells out exactly what they are.

As far as possible, the SPK code deals internally with right-handed systems, transforming into left-handed systems only to harmonize with existing astronomical practice. It is advisable in code to provide explanatory comments each and every time there is opportunity for misunderstanding.

Angles may be expressed in any range (in particular $\pm\pi$ or $0 - 2\pi$) without changing meaning as far as SPK is concerned. A TCS which had reason to be careful about this aspect, for example where dealing with cable wraps and limits, would have to condition values returned by SPK appropriately.

6.1 Vectors

Throughout SPK, vectors are right-handed. There are, for example, no cases where left-handed spherical coordinates such as $[Az, El]$ are transformed into 3-vectors using standard formulas; the angular coordinates would be converted into right-handed forms before transformation.

6.2 Spherical coordinates

Celestial coordinates in $[\alpha, \delta]$ form are naturally right-handed, and no special care is needed dealing with them.

The opposite is true of $[h, \delta]$, which is a left-handed system. The right-handed form is $[-h, \delta]$.

The other awkward case is $[Az, El]$, with the additional complication that in practice different zero points are encountered. In SPK the left-handed form is clockwise from north as seen from above, while the right-handed form is counter-clockwise from south.

SPK calls generic mount coordinates $[roll, pitch]$, a right-handed system. For equatorial mounts, zero roll is on the meridian; for altazimuth mounts, zero roll is south. In code that transforms $[roll, pitch]$ into either $[Az, El]$ or $[h, \delta]$ there must be comments spelling out exactly what is happening.

6.3 Focal plane

The focal plane $[x, y]$ system is right-handed as seen by the light approaching the focal plane being delivered from Cassegrain optics or another non-mirror-imaging optical design. This means that **when projected onto the sky** the $[x, y]$ axes are **left-handed**.

6.4 Rotator angle

Zero rotator angle is when the projection on the sky of the y -axis points at the negative mount pole, hence downwards in an altazimuth and south in an equatorial. The angle increases counter-clockwise on the sky.

7 The operational pointing model

This section deals with the chain of events that must come about if accurate corrected pointing and tracking is to be achieved, namely calibration, modeling and implementation.

For the modeling stage an offline pointing-analysis program called PMFIT is supplied with SPK, but is neither SPK-specific nor mandatory. PMFIT uses a rudimentary but quite effective fixed six-term model, but far better results will be obtained with one of the much more complete proprietary pointing-analysis packages (ideally TPOINT).

7.1 Context

Straight out of the box, SPK does little more than point a perfect telescope/mount at a celestial target. But telescopes/mounts are never perfect, and to achieve accurate pointing and tracking there needs to be a way of correcting for mechanical defects such as misalignments, flexures and gear errors. The interface through which these corrections are applied is the set of seven coefficients in SPK's *internal* pointing model structure, the argument `pm` in the function `spkVtel`.

The TCS developer must devise ways of accomplishing three things:

1. Set-piece pointing tests have to be developed, sampling the behavior of the telescope+mount at different places in the sky. See Section 7.2.
2. Suitable mathematical terms must be fitted to these samples that describe the character of the all-sky pointing. This set of formulas and amplitudes is the *operational* pointing model. See Section 7.3.
3. In the TCS pointing kernel, the internal model `pm` must be kept updated in real time so that it impersonates the operational model. See Section 7.4.

7.2 Pointing tests

Before models can be fitted and correction code written, the essential first step is to develop procedures that carry out pointing tests. The telescope must be pointed at a series of calibration targets, and records kept of, in effect, (i) the mount encoder readings that the telescope was told to point at and (ii) where in the sky it was actually pointing.

The latter, the direction of the incoming radiation, is known as “observed place”. Starting from the catalog entry for the star, it includes all the the usual astrometric transformations up to and including refraction. What is actually logged can be the observed coordinates themselves or enough information (star catalog entry, ambient air conditions, color, time) to compute them.

The pointing-test implementation must have access to a set of trustworthy calibration targets, and there are two options for these. The most straightforward is simply a list of stars

(optical/IR or radio), chosen for their brightness and isolation; acquiring each pointing sample normally involves centering on the star before logging the results. The second method, which applies only to optical/IR and requires a camera, is to make exposures at arbitrary points spread over the sky and then perform automatic astrometry to determine from the star patterns where the center of the exposure turned out to be.

The TCS implementor has to decide the order in which the pointing calibrators will be observed. The whole observable sky must be sampled, fairly evenly, and with some degree of randomness; avoiding excessive dome motion will also be a consideration. A tried-and-tested scheme is as follows:

1. Observe the calibration target which is (a) nearest where the telescope happens to be pointing and (b) sufficiently high in the sky.
2. Mark it as used.
3. Repeat from Step 1 until all visible calibrators have been used.

It is very important to log the target's observed place and the encoder demands as *absolute* quantities, sampled at precisely the same instant and independent of whatever pointing/tracking corrections the TCS is currently applying. A trap that TCS developers usually fall into is instead to log how far out the star was. Radio astronomers for some reason often make matters worse by recording azimuth + error separately from elevation + error, and not necessarily at the same instant. If this line-of-least resistance "How far out were we?" method is used, not only does the operational pointing model have to be recorded but the calculations during analysis must be identical to what the TCS was doing. There is a lot to go wrong!

In summary, the pointing samples should fulfil the following requirements:

- The observations must cover the full patrol range of the telescope and be evenly spread across the sky. They must be random and not a regular grid of any kind.
- The quantities recorded must be absolute, and not relative to the current pointing model. In the case of altazimuth mounts this will require adroit real-time handling.
- The telescope readings must be mechanical rather than celestial, while the star readings must be observed place (or the makings).
- The path must keep both axes busy to allow temporal and spatial effects to be distinguished during analysis.
- All observations must be vector ones, *i.e.* both coordinates at once.
- The observations should either be centered or astrometrically solved. Simply recording pointing errors or adjustments is not acceptable.

7.3 Modeling

Given a set of pointing samples, the next step is to model the relationship between where the telescope was told to point and where it actually ended up pointing. This can be done in a rudimentary way by the supplied offline tool PMFIT, but serious work calls for a proprietary package such as TPOINT, MaxPoint, PointXP *etc.* In all of these packages the goal is to minimize Σr^2 , where r is the error on the sky.

The recommended proprietary pointing-analysis tool is TPOINT. This offers a huge repertoire of terms (not the fixed set of just six terms that PMFIT uses), the ability to build models both interactively and automatically, formal statistical tests to identify outliers and assess term effectiveness, the ability to combine heterogeneous pointing tests for analysis, rigorous Euler-angle based transformations, and much more.⁸

7.3.1 Using PMFIT

The PMFIT program reads a file of pointing-test data from standard input, fits a basic model that minimizes the sum of squares of the sky error, and reports the fit through standard output.

The input file has the following format:

- Comment.
- Mount record: E or A for equatorial or altazimuth respectively.
- Latitude record (degrees).
- Observation records.

Each observation record contains four numbers, all of which are angles in degrees.

The first pair are the observed place, the direction from which the radiation from the pointing calibrator is arriving. The second pair are the encoder demands needed to center the telescope on the calibrator.

For the equatorial case the observations are in terms of hour angle (west positive) and declination. For the altazimuth case the observations are in terms of azimuth (north through east) and elevation.

The operational model that PMFIT produces contains only six terms, namely the two index errors, telescope/pitch non-perpendicularity, the two components of the tilt of the roll axis,

⁸Examples of TPOINT in action can be found in:

- Mangum, G. *et al.*, 2006, *Evaluation of the ALMA Prototype Antennas*, Publications of the Astronomical Society of the Pacific, **118**, 847, 1260, and
- White, E., *et al.*, 2022, *Green Bank Telescope: overview and analysis of metrology systems and pointing performance*, Astronomy & Astrophysics, **659**, A113.

and Hooke's Law flexure. Note that there is no PMFIT term that corresponds to nonperpendicularity between the two axes of the mount. This is because that and two other terms, namely the roll zero point and the collimation error, are highly correlated and consequently hard to separate. Given typical standards of engineering, it is safer to assume that the mount axes are at right angles.

For demonstration purposes, two samples of pointing-test data are provided, one equatorial and one altazimuth. In both cases the observations are authentic, complete and, apart from having been transformed into PMFIT format, unedited.

The first, `pmfit_equat.dat`, is a 39-observation data set from a 5m horseshoe-mounted telescope in the northern hemisphere:

```
5m horseshoe equatorial
E
+33.356111
  6.330566 +30.152204  6.340084 +30.170576
  0.187975 -33.082993  0.199617 -33.063655
  1.817108  +9.797865  1.830450  +9.815605
      :
      :
  1.947325 +30.766812  1.957668 +30.784955
  1.983704 +30.766812  1.994047 +30.784950
```

The second, `pmfit_altaz.dat`, is a 105-observation data set from from an 8m altazimuth telescope in the southern hemisphere:

```
8m altazimuth
A
-30.240667
264.465524  28.162130 264.433446  28.271060
259.687217  29.877367 259.655032  29.985096
259.231614  42.583144 259.202411  42.677360
      :
      :
110.331319  72.458197 110.282862  72.514584
130.940386  39.626090 130.901588  39.719032
```

The PMFIT report for the equatorial data set is:

IH:	-92.258	+/-	4.034	arcsec
ID:	-126.738	+/-	2.273	arcsec
CH:	+44.210	+/-	3.266	arcsec
ME:	+64.848	+/-	2.495	arcsec
MA:	-2.992	+/-	1.395	arcsec
TF:	+10.993	+/-	1.647	arcsec

Sky sigma = 5.780 arcsec

The coefficients IH and ID are the index errors (zero points) in hour angle and declination, CH is the “collimation error”, the nonperpendicularity between the telescope and the declination axis; ME and MA are the polar axis displacements up-down and left-right respectively; and TF is the Hooke’s Law vertical flexure. *n.b.* PMFIT uses first-order expressions for CH, ME and MA, whereas the TPOINT package uses rigorous rotations.

TPOINT automatic modeling of the same data set achieved sky σ of 1.65 arcsec from a 14-term model. The PMFIT 5.78 arcsec result is therefore only a factor of four worse, which is unusually good: a factor of 5-10 would be more typical. To illustrate the available performance without pointing corrections, fitting only the zero points gave $\sigma = 36.26$ arcsec, so for this example PMFIT achieved a factor of six improvement.

The PMFIT report for the altazimuth data set is:

IA:	-142.148	+/-	4.378	arcsec
IE:	-117.293	+/-	3.016	arcsec
CA:	+19.843	+/-	3.022	arcsec
AN:	-23.758	+/-	0.965	arcsec
AW:	+19.261	+/-	0.960	arcsec
TF:	+279.752	+/-	4.361	arcsec

Sky sigma = 8.349 arcsec

IA and IE are the azimuth and elevation index errors; CA is the nonperpendicularity between the telescope and the elevation axis; AN and AW are the azimuth axis tilts north and west respectively; and TF is the Hooke’s Law vertical flexure. *n.b.* PMFIT uses first-order expressions for CA, AN and AW, whereas the TPOINT package uses rigorous rotations.

TPOINT automatic modeling of the same data set achieved sky σ of 1.01 arcsec from an 18-term model. The PMFIT 8.36 arcsec result is therefore a factor of eight worse, a not untypical result. Fitting zero points alone gave $\sigma = 60.44$ arcsec, so for this example PMFIT achieved a factor of seven improvement.

7.4 Implementing the model

With a mathematical model of the pointing characteristics adopted, the final step is to implement new code in the `spkVtel` function to evaluate the operational model in real time. This involves updating the data members of the `pm` structure at some appropriate rate.

For a few basic terms there is a one-to-one correspondence between the operational and internal models. In other cases the terms in the operational model will turn into contributions to two or more internal model terms. The documentation for the pointing-analysis tool of choice should provide the mathematical expressions that enable these contributions to be formulated.

In less sophisticated analysis tools, all the operational model terms are presented simply as contributions to the index errors, using first-order spherical trigonometry expressions. In that scenario only two of the internal model's coefficients will need to be populated, and the remaining five left set to zero. This simple-minded scheme is not ideal as it will fail to exploit SPK's rigor, so that both pointing and field orientation predictions will progressively degrade as the pointing coefficients increase in size.

Irrespective of what pointing-analysis tool is chosen, coding the `spkVtel` enhancements is bound to be messy and error-prone. Fortunately, the code will be very easy to debug, using what is called a “dummy pointing test”. This is where a pointing test is run but the data logging is contrived to correspond to no pointing errors: the corrected pointing that the TCS is calculating is already producing a perfect result. Fictitious stars can of course be used, and there is no need for darkness; indeed there is no need to have the telescope moving at all. The subsequent pointing analysis should give exactly the same terms and coefficients that were generated by the analysis tool after an actual pointing test. With luck, any differences will merely be sign errors or missing terms, hence easy to spot. However, it will usually be necessary to deduce what is going wrong from the look of the pointing residuals. If that fails, experimenting with one term at a time is the way forward.

The following lines of code in the demonstrators `altaz.c` and `equat.c` show what is involved. Real code would of course have amplitudes passed in from outside in some way, not the hardcoded numerical values used here.

Using the term amplitudes from the above PMFIT examples, the `equat.c` code would be:

```
pm.pia = +92.258 * AS2R;          /* -IH */
pm.pib = -126.738 * AS2R;        /* +ID */
pm.pvd = +44.210 * AS2R * cos(tare); /* +TF */
pm.pca = -64.848 * AS2R;        /* -CH */
pm.pnp = 0.0 * AS2R;            /* -NP */
pm.paw = +2.992 * AS2R;         /* -MA */
pm.pan = +10.993 * AS2R;        /* +ME */
```

...and the `altaz.c` code would be:

```

pm.pia = -142.148 * AS2R;          /* +IA */
pm.pib = -117.293 * AS2R;          /* +IE */
pm.pvd = +19.843 * AS2R * cos(tare); /* +TF */
pm.pca = -23.758 * AS2R;          /* +CA */
pm.pnp = 0.0 * AS2R;              /* +NPAE */
pm.paw = +19.261 * AS2R;          /* +AW */
pm.pan = +279.752 * AS2R;          /* +AN */

```

In both cases, `cos(tare)` is the cosine of the target elevation (*i.e.* the sine of zenith distance) and there is no NPAE/NP component for the reasons given earlier.

8 Files

The following lists all the SPK files and the purpose of each one:

<code>altaz.c</code>	C source for the <code>ALTAZ</code> demonstrator
<code>astr.c</code>	C source for the <code>spkAstr</code> function
<code>ctar.c</code>	C source for the <code>spkCtar</code> function
<code>equat.c</code>	C source for the <code>EQUAT</code> demonstrator
<code>pmfit.c</code>	C source for the <code>PMFIT</code> model-fitting program
<code>pmfit_altaz.dat</code>	simulated pointing-test data, altazimuth
<code>pmfit_equat.dat</code>	simulated pointing-test data, equatorial
<code>point.c</code>	C source for the <code>POINT</code> demonstrator
<code>s2e.c</code>	C source for the function called by <code>point.c</code>
<code>spk.h</code>	C source for the <code>spk.h</code> header file
<code>spk.pdf</code>	documentation PDF
<code>spk.tex</code>	documentation \LaTeX source
<code>vtel.c</code>	C source for the <code>spkVtel</code> function