Telescope Pointing Machine Specification Version 2.2

Jeffrey W Percival
Space Astronomy Laboratory
University of Wisconsin

December 28, 2005

Contents

List of Figures

List of Tables

1 Introduction

This document specifies the Telescope Pointing Machine (TPM) software. This is a table-driven software state machine that produces and reduces coordinates for the purposes of pointing a telescope, but which is generally applicable to any astronomical coordinate application. We emphasize the pointing of a telescope to underscore the fact that the Telescope Pointing Machine is a fast, compact, rigorous, and portable implementation that allows the user to control and tune the computational load in real time environments such as telescope control systems. It contains no vendor or platform dependencies, however, and is therefore suitable for any astronomical coordinate application.

The TPM can transform any set of coordinates (positions *and* velocities) between any pair of 21 different states, which include galactic, ecliptic, equatorial and topocentric systems, in either direction, with a single subroutine call.

The TPM uses rigorous vector/matrix methods for its transformations, avoiding spherical trigonometry and its associated singularity problems. The transformation between any two states is reversible to within 4 milliarcseconds, and all transformations agree with Starlink's SLALIB, considered to be definitive, to within 5 milliarcseconds, with the differences being attributable to floating point precision limits. SLALIB comparison programs are included in the TPM distribution. The TPM computes apparent places of stars in agreement with the United States Naval Observatory's NOVAS code, used to produce the Astronomical Almanac, to within 10 milliarcseconds. This error is comparable to the precision of the data provided by the USNO, and so the two codes are indistinguishable at this level of error.

The paragraphs below establish a context for subsequent sections. They define some terms, lay out the big picture, and describe some of the special features of this software.

In this paper, we often refer to the Explanatory Supplement to the Astronomical Almanac (1992), Yallop et al. (1989, AJ, 97, 274), Kaplan et al. (1989, AJ, 97, 1197), and Aoki et al. (1983, Astronomy and Astrophysics, 128, 263). We will refer to these sources as ES, Yallop, Kaplan, and Aoki.

1.1 Vectors vs. Spherical Trigonometry

There has been a general trend in recent years in which algorithms are expressed in vector and matrix notation instead of using spherical trigonometry. In some cases, the trigonometric expression is mathematically equivalent to the vector expression, but in other cases it may only be, say, a first or higher order approximation. Two examples will suffice to make the point.

First, consider the aberration of light. Classical aberration is given in the ES (Equation 3.253-1) as

$$\begin{aligned}\cos\delta\Delta\alpha &= -\frac{\dot{X}}{c}\sin\alpha+\frac{\dot{Y}}{c}\cos\alpha \\ &\quad +\frac{1}{c^2}(\dot{X}\sin\alpha-\dot{Y}\cos\alpha)(\dot{X}\cos\alpha+\dot{Y}\sin\alpha)\sec\delta+\ldots \qquad (1)\\ \Delta\delta &= -\frac{\dot{X}}{c}\cos\alpha\sin\delta-\frac{\dot{Y}}{c}\sin\alpha\sin\delta+\frac{\dot{Z}}{c}\cos\delta \\ &\quad -\frac{1}{2c^2}(\dot{X}\sin\alpha-\dot{Y}\cos\alpha)^2\tan\delta \\ &\quad +\frac{1}{c^2}(\dot{X}\cos\delta\cos\alpha+\dot{Y}\cos\delta\sin\alpha+\dot{Z}\sin\delta) \\ &\quad \times(\dot{X}\sin\delta\cos\alpha+\dot{Y}\sin\delta\sin\alpha-\dot{Z}\cos\delta)+\ldots \qquad (2)\end{aligned}$$

where $\dot{X}$, $\dot{Y}$, and $\dot{Z}$ represent the observer's barycentric velocity.

Expressed as a vector equation, and accurate to the milliarcsecond level, aberration is given in the ES (Equation 3.255-2) as

$$\vec{p_2}=\vec{p_1}+|\vec{p_1}|\frac{\vec{v_1}}{c} \qquad (3)$$

where $\vec{p}$ and $\vec{v}$ are the position and velocity vectors, respectively.

Equation 3 is easier to look at than Equations 1 and 2, it is easier to implement, faster to compute, and retains much more physical meaning: the effect is largest when $\vec{p_1}$ and $\vec{v_1}$ are perpendicular, and is proportional to the observer's speed. Also note that the classical expression contains polar singularities, but the vector expression has no such restrictions.

Second, consider the reduction for parallax from barycentric place to geocentric place (ES, Equation 3.243-1):

$$\alpha_2=\alpha_1+\pi(X\sin\alpha-Y\cos\alpha)/(15\cos\delta) \qquad (4)$$

and

$$\delta_2 = \delta_1 + \pi(X \cos\alpha \sin\delta + Y \sin\alpha \sin\delta - Z \cos\delta) \tag{5}$$

where π is the parallax in arcseconds and (X, Y, Z) are the Earth's barycentric coordinates.

In vector form, we have (ES Equation 3.325-1)

$$\vec{p_2} = \vec{p_1} - \vec{E} \tag{6}$$

where $\vec{p_1}$ is the barycentric target vector, $\vec{p_2}$ is the geocentric target vector, and $\vec{E}$ is the barycentric position vector of the Earth. Again, simplicity, speed, and clarity prevail.

The TPM system uses vector and matrix formulations throughout. No conversions are done back and forth between Cartesian and spherical notation. In addition, this software system uses *state* vectors, not merely *position* vectors.

1.2 What is a State Vector?

The TPM transforms *state vectors* from one state, or reference frame, to another. A *state vector* is a 6-element vector whose elements specify the target's position and velocity. The state vector can be expressed in either Cartesian $(x, y, z, \dot{x}, \dot{y}, \dot{z})$ or spherical $(r, \alpha, \delta, \dot{r}, \dot{\alpha}, \dot{\delta})$ form, with subroutines provided for switching from one to the other. The position and velocity are real, 3-dimensional true-length vectors, as opposed to unit vectors comprising direction cosines, for example. The state vector is expressed in units of AU and AU/day, for compatibility with the JPL DE-200 Planetary Ephemeris.

Using real 3-D positions, rather than unit vectors, simplifies many pointing calculations. For example, applying barycentric parallax (moving from barycentric to geocentric coordinates) is done merely by subtracting the barycentric position vector of the Earth (three subtractions). If a target vector were kept as a unit vector, the Earth's position vector would have to be scaled down before the subtraction, and then the target vector would have to be renormalized to retain its unit length after the subtraction. This is extra work that doesn't have to be done with true-length target vectors.

Carrying along the velocity vector with the position vector, and implementing the algorithms as 6-dimensional phase-space problems instead of the usual 3-dimensional position problems is another big improvement over the usual unit-vector, position-only methods. If the simple vector subtraction given above for annual parallax is performed in 6-D, then you end up not only with the geocentric position, but the geocentric velocity as well. This velocity is ready-made for, say, calculating the aberration of light or for measuring orbitally-induced spectroscopic Doppler shifts seen at the telescope. Working with 6-D state vectors means that velocity vector is transformed wherever appropriate; proper motions in equatorial coordinates end up as proper motions in azimuth and elevation, for example.

There is something even subtler you get by using true-length 6-D state vectors. When applying proper motion, it is usually sufficient, but not rigorously correct, to use

$$\alpha_2 = \alpha_1 + \dot{\alpha}\Delta T \tag{7}$$

and

$$\delta_2 = \delta_1 + \dot{\delta}\Delta T. \tag{8}$$

Why isn't this rigorous? It ignores the radial velocity, and when the full space motion $(\dot{r}, \dot{\alpha}, \dot{\delta})$ is applied to a position *it changes the radial velocity and parallax* (see Equations 40 and 43). In 6-D, if one applies space motion according to

$$\vec{p_2} = \vec{p_1} + \vec{v_1}\Delta T \tag{9}$$

and then rigorously converts $\vec{p_2}$ back into a spherical representation, using Equations 40-45, the radial velocity and parallax will be correct.

Finally, we should point out that while many pointing codes assume that space motion is small and always represents stellar proper motion, we make no assumptions about the length of the velocity vector. The velocity vector can represent the motion of solar system objects such as asteroids or comets, and the fact that these are not linear in an equatorial frame is mitigated by the fact that they can be expressed in an ecliptic frame, where the linear approximation is much better.

1.3 What is a State Machine?

Computer programs are typically written as a sequence of subroutine calls. First do A, then B, then C, and so on. The subroutines are locked in place in the program, and the sequence of operations is fixed and immutable. A software state machine, on the other hand, is written as a loop, and at the beginning of each trip through the loop, the program asks a scheduler what to do next. In effect, the program says "OK, here's what I just did, and here's what I'm trying to do, what should I do next?" The execution is *dynamic* in the sense that the scheduler can assign subroutines on the fly, depending on circumstances that can change at any time.

In the case of the TPM, there are only two circumstances that matter: where in the calculation the program currently is and where it's trying to go. The scheduler is implemented in the form of a lookup table. This is a two-dimensional grid of numbers whose rows are indexed by the current state of the program, and whose columns are indexed by the desired end state. For example, if the current state is "galactic coordinates" and the desired state is "topocentric observed azimuth and elevation," the lookup table will yield two data: the next thing to do (convert to FK4 B1950 equatorial), and the state that results (FK4 B1950 equatorial). On the next trip through the loop, the scheduler will see (FK4 equatorial, topocentric observed) and will dictate "convert from FK4 to FK5" as the next thing to do. The program will keep looping until the state table says "you're done," at which point the end-to-end algorithm is complete.

All the knowledge of the program is encoded into the state table, and the state table embodies *all* the relationships between *all possible states*.

1.4 State Diagram and State Table

The Telescope Pointing Machine handles all the states and transitions shown in Figure 1. Each transition is implemented only once, in a single place, with the required transition being given as a function of the current state and the destination state. The algorithms for each state transition are described in detail below.

The state table is given in Tables 1 and 2. The rows are prefixed with the current state, and the columns are headed with the destination state. Each table entry is a pair of numbers that denote the required transition and the new state resulting from that transition. Note that the transition alone does not define the resulting state, because a given transition can appear more than once in Figure 1, connecting different pairs of states.

1.5 Programming Advantages of the Telescope Pointing Machine

The two biggest advantages of the state machine approach is that *all* possible transitions have been coded into the state table, and only one subroutine call is needed to execute any coordinate transformation.

The first advantage means that the programmer doesn't need to know the algorithmic details of coding up a coordinate transformation. For example, does nutation precede or follow precession? Is aberration done before or after gravitational deflection of light? Do galactic coordinates map into FK4 or FK5 equatorial coordinates? While it is true that procedural toolkits can mimic this encapsulation by preparing certain "super routines" that group together commonly needed subroutines, the state machine implementation embodies the correct sequence of events for *all possible transformations.*

The second advantage means that the programmer doesn't need to know the programmatic details of coding up a coordinate transformation. Is the precession routine called with time arguments of Julian Dates, Modified Julian Dates, or floating point year values? If years, are they tropical years or Julian years? The state machine makes all the lower-level subroutine calls, and the subroutine interfaces are internal to the state machine. This provides an additional related advantage: the underlying code base is easier to change, if needed, because the programmer's interface is fixed and constant.

Another advantage of the table-driven state machine is the easy way in which new states can be added. The TPM programmer merely needs to specify the new state, connect it to the computationally nearest existing state by providing a forward and back calculation, and update the state table. The new state is now connected to *all* existing states, and is accessible without changing the subroutine interface at all!

A final thing to realize about this implementation is that it is a multiple-pass state machine, not just a table lookup. For example, one might have considered indexing a state table by the start and end states, with the state table giving a *complete* specification of the desired transformation. At each intersection we could have just inserted the whole laundry list of transformations required to get from the start state to the end state. This is a bad idea. Consider adding a state: a new recipe would have to be concocted for each combination of the new state and existing states, with the new state being either a start or destination. The programmer would have to understand *all* transformations to get this right.

In this implementation, the state table just gives the *next* thing to do, not *all* things to do. The state table embodies the relationships between transformations. The state machine keeps looping through, doing one transformation at a time, until it discovers it is done. To add a new state, it is only necessary to connect it to the *nearest* state, not to all states, reducing the breadth of knowledge required by the programmer as well as reducing the potential of introducing errors.

2 State Data

The transformation of coordinates between any two states in Fig. 1 is driven by 30 numerical quantities. Twelve of these are *independent* variables, chosen by the user. The remaining 18 are *dependent* variables, derived in due course from the independent variables.

2.1 Independent State Data

The independent variables (and their units) are:

- DAT (seconds of time): the accumulated number of leap seconds, currently 29 on 18 November 1995.
- UTC (Julian date): coordinated universal time, formerly GMT.
- DUT (seconds of time): the current value of the difference UT1 - UTC, varying between -0.3 and +0.7 seconds.
- Polar motion X,Y (radians): two angles describing the current wandering of the instantaneous rotation axis of the Earth. Polar motion affects pointing at about the 0.3" level.
- Geodetic east longitude (radians): the mean longitude of the observer, reckoned *negatively* for the Western hemisphere.
- Geodetic mean north latitude (radians): the observer's north latitude.
- Altitude (meters): the observer's height above mean sea level.
- Temperature (Kelvins): the atmospheric temperature at the observer.
- Pressure (millibars): the atmospheric pressure at the observer.
- Relative humidity (fraction, 0-1): the atmospheric relative humidity at the observer.
- Observing wavelength (nanometers): the wavelength of light being observed, used for the refraction correction.

2.2 Dependent (derived) State Data

The dependent variables are:

- TAI (Julian date): International Atomic Time, the time measured by an ensemble of atomic standards.
- TDT (Julian date): Terrestrial Dynamical Time, the time used as the independent variable in geocentric ephemerides.
- TDB (Julian date): Barycentric Dynamical Time, the time used as the independent variable in solar system barycentric ephemerides.
- UT1 (Julian date): The rotational time scale of the Earth, corrected for polar motion.
- GMST (radians): Greenwich Mean Sidereal Time.
- GAST (radians): Greenwich Apparent Sidereal Time.
- LAST (radians): Local Apparent Sidereal Time.
- Precession matrix: a 6x6 matrix used to precess coordinates from J2000 to date in the inertial FK5 reference frame.
- Nutation matrix: a 3x3 matrix used to account for the small lunar and solar torques on the Earth.
- Obliquity of the ecliptic (radians): the instantaneous angle between the ecliptic and mean equatorial planes.
- Nutation in obliquity (radians): the instantaneous angle between the mean and true equatorial planes.
- Nutation in longitude (radians): the instantaneous angle between the mean and true location of the vernal equinox, measured along the ecliptic plane.
- Refraction coefficients A and B (arcseconds): these two angles are computed from the basic atmospheric data, and parameterize a simple model of the elevation dependence of atmospheric refraction.

- Earth's barycentric and heliocentric state vectors (AU and AU/day): the state vectors of the Earth, referred to the mean equator and equinox of J2000.
- Observer's mean earth-fixed state vector (AU and AU/day): the geocentric state vector of the observer, expressed in a reference frame rotating with the earth, referred to the mean geodetic pole.
- Observer's true earth-fixed state vector (AU and AU/day): the geocentric state vector of the observer, expressed in a reference frame rotating with the earth, referred to the true geodetic pole. Polar motion is included here.
- Observer's space-fixed state vector (AU and AU/day): the geocentric state vector of the observer expressed in a space-fixed reference frame, referred to the mean equator and equinox of J2000.

2.3 Relationships between State Data

Figure 2 shows these 30 data in the form of a family tree, highlighting the relationships between them. In this figure, the independent variables lie across the top of the tree, in the first generation. The dependent variables flow down from the independent ones, with the lines tracing the dependencies.

The 18 dependent variables fall into three broad categories: those that change quickly, on time scales smaller than a second, those that change on time scales of minutes to hours, and those that change on time scales of days to years. They fall into a different three categories as well: those that are cheap to compute (a few additions and multiplications), those that are moderately expensive to compute (a dozen or so trigonometric evaluations), and those that are expensive to compute (numerical integrations, thousands of trigonometric evaluations, and iterative procedures).

Nature is kind to us in this area, because the most expensive things to compute generally change on the longer time scales (like the nutation in longitude, for example), so that they do not need to be computed as often as the quickly changing things (like the sidereal time, for example).

Figure 2 categorizes each item by computational cost, and groups together items of similar time scale of change. The TPM system provides a simple subroutine called `tpm_data()` for initializing the state data and for computing the fast, medium, slow, and refraction data separately or together.

Figure 3 shows the relationship between the various civil, dynamical, and rotational times. The TPM subroutine library provides subroutines and preprocessor macros to execute all the transitions shown in Figure 3.

Figure 4 shows the C structure declaration for the state data.

3 Notes on Equinox and Epoch

The meanings of *equinox* and *epoch* are perennial sources of confusion for the non-specialist. In order to be absolutely clear on this subject, we will demonstrate the relationship of the relevant dates used in pointing by treating the problem in its simplest form. Define the current position $\vec{r}$ of a moving target in some coordinate system to be

$$\vec{p} = \vec{p_0} + \vec{v}(t - ep) \tag{10}$$

where $\vec{p_0}$ is the position at time *ep*, and $\vec{v}$ is the velocity of the target. In this equation, time t is the desired time of observation (i.e. "now" when pointing a telescope). Time *ep* is the *epoch*. Now consider a second coordinate system, rotated with respect to the first by some time-varying angle θ. In other words,

$$\theta = \theta_0 + \dot{\theta}(t - eq) \tag{11}$$

where θ_0 is the angle at time *eq* and $\dot{\theta}$ is the angular rate of rotation. Time *eq* is the equinox. If $R(\theta)$ represents the coordinate rotation, and $\vec{q}$ the position in the rotated frame, then

$$q = R(\theta_0 + \dot{\theta}(t - eq)) * (\vec{p_0} + \vec{v}(t - ep)). \tag{12}$$

Note that we have *three* times here: the current time t, the target motion's reference time *ep*, and the coordinate frame's reference time, *eq*. All of these can differ from each other. For example, when using catalog entries to point a telescope, *eq* will often be the same as *ep* (say, J2000), but t will be *now.* At the turn of the century, all three would be equal for an instant. When viewing a comet, on the other hand, an ephemeris will often be generated for a recent epoch (say, last night or a week ago), but in the reference frame of J2000. In this case, t will *nearly* equal *ep*, but not quite, and both will differ markedly from *eq*. All three will differ if one extracts positions from a reference image exposed in, say, 1990, but using J2000 reference positions, for an object to be observed in 2008.

4 Notes on Precession

There are different equinoxes involved in precession in various parts of Fig. 1. A user-supplied equinox (*eq* in the C code) is used in T01 and T02. *eq* is the starting time in the forward directions (+T01 and +T02), and the ending time in the reverse direction (-T01 and -T02).

In transition T10, the starting time for the precession is J2000, and the ending time is “now”, specified by the value of UTC in the state data structure. These are of course reversed in -T10.

Note that some coordinate transformations require two invocations of TPM. Consider precession from one equinox to a different equinox (S02 to S02). There is only one equinox argument to the tpm routine, and in addition, asking for start and end states to be both S02 will cause TPM to return without having done anything.

In this example, the user would do S02 to S06 (with the starting equinox) in the first invocation, leaving the state vector temporarily referred to J2000, and then do S06 to S02 (with the ending equinox) in the second invocation.

5 Notes on Proper Motion

Proper motion is the trickiest subject in using TPM. First note that TPM never actually *applies* proper motion to the position. The user supplies a position as a 3-D cartesian vector in AU, and the velocity as another 3-D vector in AU/day. TPM performs its coordinate transformations on *both* vectors. Rather than *applying* the proper motion, it *transforms* it from one frame to another.

But if proper motion is never applied, why does the user supply an epoch when calling the tpm subroutine? The answer is that T05 (FK4/FK5) requires an application of space motion. The FK4 equinox correction must be applied to the position referred to the epoch and equinox of 1984 Jan 1.

In applying +T05, TPM applies proper motion from the user-supplied epoch value to B1950. The FK4-to-FK5 correction then applies proper motion from B1950 to 1984 Jan 1.0, applies the FK4 equinox correction, then applies more proper motion from 1984 to J2000. In keeping with the intent of not actually applying proper motion to the position vector, TPM then *removes* proper motion from the position, going from a J2000 position back to the position at the user-supplied epoch.

The net effect of +T05 is to transform a position referred to B1950 at a user-supplied epoch, into a position referred to J2000, *but at the same user-supplied epoch*.

In -T05, the user's position is brought from the user-supplied epoch to J2000, sent back to 1984, then to B1950, then back to the user-supplied epoch.

6 Notes on Apparent Places

Algorithms for computing the apparent (geocentric) places of stars exist in different forms, some using spherical trigonometry while others use vectors and matrices. Some exist as approximations, while others are exact. These variations have an historical heritage. Many derive from times when computing was tedious or expensive. For example, the traditional splitting up of aberration into annual (geocentric) and diurnal (topocentric) components was clearly intended to ease the burden of computation. The annual component could be done once, along with precession and nutation, on a fairly coarse time grid and tabulated in advance (as was actually done for apparent places). The site-specific diurnal correction could be ignored for most applications (e.g. pointing telescopes) or easily applied for higher precision work.

In a modern vector-based system using state vectors, there is less of an argument for making these sorts of distinctions. There are even advantages in abandoning them: not all observers are even on the Earth any more, and if we treat, say, aberration as a single effect (as does the TPM system), then we can consider "apparent places" for atypical observers such as orbiting spacecraft.

The TPM can compute the traditional geocentric apparent places, but it also defines a "topocentric" apparent place that uses the observer's net barycentric position and velocity in computing parallax and aberration. The "topocentric" label is slightly misleading in this context, because the TPM user can supply *any* state vector, for an observer anywhere in the solar system.

7 Notes on State Transitions

The following subsections present detailed notes on each state transition. We describe the transformations using mathematical notation instead of, say, subroutine interface descriptions. We can do this because the state machine calls the subroutines, not the user's program, so the programmer never needs to know the calling interface. This is a vast improvement over procedural toolkits, where many subroutines must be organized by the user with careful attention paid to the order and detailed nature of each argument of each subroutine.

We make extensive use of vector and matrix notation here. The standard rotation matrices are taken from Yallop, and are reproduced here:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{13}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{14}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

These orthonormal matrices have the property that

$$R_i^{-1}(\theta) = R_i^T(\theta) \tag{16}$$

We further define, following Yallop,

$$Q_i(\theta, \dot\theta) = \begin{bmatrix} R_i(\theta) & 0 \\ \dot R_i(\theta) & R_i(\theta) \end{bmatrix} \tag{17}$$

Q represents a 6-space rotation that operates on both the position and velocity components of a state vector, and can handle the transformation from non-inertial to inertial reference frames through the time derivative term $\dot R_i(\theta)$.

In the following sections, $\vec{s}$ represents a full 6-space state vector, $\vec{p}$ represents a 3-space position vector, and $\vec{v}$ represents a 3-space velocity vector.

7.1 T01: FK4 Precession to B1950

This transition precesses the target state vector from the mean FK4 equator and equinox of date to that of B1950 according to

$$\vec{s_2} = Q_z(-z, 0)Q_y(\theta, 0)Q_z(-\zeta, 0)\vec{s_1}. \tag{18}$$

The angles (z, θ, ζ) are those of Kinoshita (see Aoki).

7.2 T02: FK5 Precession to J2000

This transition precesses the target state vector from the mean FK5 equator and equinox of date to that of J2000 according to Equation 18.

The angles (z, θ, ζ) are those adopted by the IAU (ES p. 104 or Yallop).

7.3 T03: IAU 1980 Ecliptic to FK5 Equatorial

This is a simple rotation about the X-axis (through the vernal equinox) to remove the obliquity of the ecliptic:

$$\vec{s_2} = Q_x(-\epsilon, 0)\vec{s_1} \tag{19}$$

where ϵ is the mean obliquity of the ecliptic of date (ES p. 114, Equation 3.222-1).

The existence of this state, combined with the ability to specify the velocity vector with values much larger than typical stellar proper motions, allows a very powerful method for tracking solar system objects such as asteroids, comets, and planets. These motions may be much more linear in the Ecliptic frame than in the equatorial frame, thereby ensuring accurate positions for a longer elapsed time than if the motions were given as rates of change in Right Ascension and Declination.

7.4 T04: IAU 1958 Galactic to FK4 B1950

The galactic pole is at $(\alpha, \delta) = (192.25°, 27.4°)$, and the longitude λ of the ascending node of the galactic plane on the equator is 33°.

The rotational transformation is

$$\vec{s_2} = Q_z(-\alpha, 0)Q_y(-(90° - \delta), 0)Q_z(-(90° - \lambda), 0)\vec{s_1}. \tag{20}$$

The E-terms of aberration are added after the rotation to equatorial coordinates, and are subtracted before the inverse rotation to galactic coordinates.

7.5 T05: FK4 B1950 to FK5 J2000

This is the subtle transformation between the rotating FK4 and inertial FK5 coordinate systems. The treatment is exactly that of Yallop and the ES, except that where Yallop uses Andoyer's precession, we follow the ES and use Kinoshita's.

We require additional modifications to the 6x6 matrix given in the ES, because the ES uses a hybrid Cartesian/spherical treatment, with a unit vector for position and a velocity vector whose units are in km/s and arcseconds per century.

Our 6-space vectors are simpler, giving the position and velocity in units of AU and AU/day, and so we require fewer numerical conversions than the standard treatment.

Specifically, we require these changes:

- The space-motion part of the matrix must be scaled by $1/(206265{*}C_B)$, where C_B is the length of the tropical century in days, 36524.21987817305. This happens automatically because our precession angle routines return radians and radians/day.

- The rotational term in the precession matrix must be scaled by $(206265{*}C_B)$. This happens automatically because our precession angle routines return radians and radians/day.

- The proper motions do not have to be scaled by C_J/C_B, where C_J is the length of the Julian century in days, 36525, because we scale them to days as part of the vector setup.

- In removing the E-terms, we must save and restore the true lengths of $\vec{p}$ and $\vec{v}$.

With these modifications, the transition consists of steps (c) and (d) from the ES p. 185. Note that with our 6-space treatment, steps (a), (b), and (e) are not required.

The TPM software includes test programs that reproduce the Yallop and ES matrices, as well as the modified versions required here. See Appendix A for details.

7.6 T06: Heliocentric Parallax

The geocentric target state vector is derived from the heliocentric target state vector by subtracting the Earth's heliocentric state vector:

$$\vec{s_2} = \vec{s_1} - \vec{E} \tag{21}$$

where $\vec{E}$ is the Earth's heliocentric state vector. $\vec{E}$ is computed by a subroutine derived from Stumpff (Astronomy and Astrophysics Supplement Series, 1980, 41, 1) by Patrick Wallace. The maximum errors, relative to the JPL DE 96 ephemeris, are 4.2×10^{-4} km/s for velocities, 1.1×10^{-5} AU (1650 km) for heliocentric coordinates, and 4.6×10^{-5} AU (6900 km) for barycentric coordinates.

Note that both the target's position *and* velocity vectors are modified in this transition. The resulting state vector gives the geocentric target state vector.

7.7 T07: Geocentric Parallax

The topocentric target state vector is derived from the geocentric target state vector by subtracting the observer's geocentric state vector:

$$\vec{s_2} = \vec{s_1} - \vec{O} \tag{22}$$

where $\vec{O}$ is the observer's geocentric state vector. $\vec{O}$ is computed according to Equations 3.351-1 and 3.351-2 in the ES, and $\dot{\vec{O}} = \omega \hat{z} \wedge \vec{O}$ where ω is the standard (Aoki) value of the angular velocity of the Earth.

Note that both the target's position *and* velocity vectors are modified in this transition. The resulting state vector gives the topocentric target state vector.

7.8 T08: Light Deflection

We implement Equation 14 from Kaplan, reproduced here:

$$\vec{p_2} = \vec{p_1} + \frac{g_1}{g_2}\left[(\vec{p_1} \cdot \hat{q})\hat{e} - (\vec{p_1} \cdot \hat{e})\hat{q}\right] \tag{23}$$

where $\hat{q}$ is the heliocentric target vector and $\hat{e}$ is the heliocentric earth vector, both of unit length, and g_1 and g_2 are the dimensionless scalar quantities defined in Kaplan's Equation 13. We differ from Kaplan in using $\vec{p_1}$ instead of $\hat{p_1}$. Following SLALIB, we limit the value of g_2 to 10^{-5}, representing a 922 arcsecond disk centered on the sun.

7.9 T09: Aberration

We use Kaplan's Equation 18, which ignores relativistic terms of order 1 milliarcsecond:

$$\vec{p_2} = \vec{p_1} + |\vec{p_1}|\frac{\vec{v_1}}{c} \tag{24}$$

where $\vec{p_1}$ and $\vec{v_1}$ are the barycentric position and velocity of the observer.

Note that we do not split aberration up into annual and diurnal contributions. Rather, we compute the effect using the total barycentric velocity of the observer. This transition is suitable for any observer in the solar system, providing the observer's state vector is properly computed.

7.10 T10: Precession from FK5 J2000 to Date

This transition precesses the target state vector from the mean FK5 equator and equinox of J2000 to that of date according to Equation 18.

The angles (z, θ, ζ) are those adopted by the IAU (ES p. 104 or Yallop).

7.11 T11: Nutation

Nutation is handled by rotating from the mean equator into the plane of the ecliptic, doing the nutation, then rotating back to the true equator:

$$\vec{s_2} = Q_x^{-1}(\epsilon + \Delta\epsilon, 0)Q_z(-\phi, 0)Q_x(\epsilon, 0)\vec{s_1} \tag{25}$$

where ϵ is the mean obliquity of the ecliptic, $\Delta\epsilon$ is the nutation in obliquity of the ecliptic, and ϕ is the nutation in longitude.

The nutations ϕ and $\Delta\epsilon$ are computed using the 108-term trigonometric series of Kaplan (USNO circular 163, 1981) with the corrections from the ES p. 116, including the planetary terms of Vondrak, ES, p. 118-119. The mean obliquity ϵ is from ES p. 114, Equation 3.222-1.

7.12 T12: Earth's Rotation

For this transition we rotate the target vector by the Greenwich Apparent Sidereal Time θ, compensate for the polar motion angles x and y, and then rotate by the observer's mean geodetic longitude:

$$\vec{s_2} = Q_z(\lambda_m, 0) Q_y(-x, 0) Q_x(-y, 0) Q_z(\theta, 0) \vec{s_1} \tag{26}$$

Finally, we switch from a right-handed coordinate system to a left-handed one by negating the Y-components of position and velocity.

We should point out a misleading, if not erroneous, statement in the ES concerning polar motion. On p. 140, polar motion is properly expressed in Equation 3.27-1 as follows:

$$\vec{r} = R_y(-x) R_x(-y) \vec{r_0}. \tag{27}$$

The next paragraph states that polar motion can be alternatively expressed as a variation in longitude λ and latitude ϕ as follows (Equation 3.27-3):

$$\Delta\lambda = (x \sin\lambda_m + y \cos\lambda_m) \tan\phi_m \tag{28}$$

and

$$\Delta\phi = x \cos\lambda_m - y \sin\lambda_m. \tag{29}$$

This is clearly incomplete, as the former expression is a two-angle rotation, while the latter is just a slight relocation of the observer. What is really meant by Equation 3.27-1, we believe, is that the first order departures from mean geodetic coordinates are useful in determining the circumstances of a star *in transit*, but not generally in determining its direction when not on the meridian. That is, $\Delta\phi$ gives the error due to polar motion of the zenith distance at transit, and $\Delta\lambda$ gives the variation in the sidereal time of transit (and therefore is used to deduce UT1 from UT0), but these two data do *not* tell how the *direction* of a star is affected by polar motion when it is not on the meridian.

The rigorous Equation 3.27-1 (our Equation 27) is, of course, the form implemented in the TPM.

7.13 T13: (HA, Dec) to (Az, El)

This is a rotation of $(90° - \textit{latitude})$ in the plane of the meridian, followed by a polar rotation of 180°:

$$\vec{s_2} = Q_z(180°, 0)Q_y(90° - \phi_m, 0)\vec{s_1}. \tag{30}$$

We use the mean geodetic latitude ϕ_m because the effects of polar motion are handled in transition T12.

7.14 T14: Refraction

The refraction procedure follows that of Patrick Wallace's SLALIB routines. Two rigorous numerical integrations are performed through a model atmosphere (see ES section 3.28) at different zenith distances, and the resulting refractions are used to derive A and B in the approximation

$$z = z_o + A\tan(z_o) + B\tan^3(z_o) \tag{31}$$

where z is the zenith distance in a vacuum and z_o is the refracted zenith distance.

The state machine actually only computes the approximation; the derivation of A and B is done in the refraction section of `tpm_data()`. The numerical integrations are done using the very efficient Romberg integration from Numerical Recipes.

7.15 T15: WHAM Coordinate System

This is built in for the WHAM telescope, which is an alt-az telescope tipped 90° to the north. It is a left-handed Cartesian system whose z-axis points to the northern horizon, and whose x-axis points straight up, toward the zenith. The y-axis points to the western horizon. The (0,0) point is straight up. The "azimuth" angle is referred to as siderostat longitude, and the "elevation" angle is referred to as siderostat latitude.

The transformation between the topocentric observed az/el and WHAM frame is

$$\vec{s_2} = Q_z(180^\circ, 0)Q_y(90^\circ, 0)\vec{s_1} \tag{32}$$

8 Speed and Accuracy

The TPM software includes programs for measuring execution times, internal accuracy, accuracy compared to the definitive procedural toolkit, Starlink's SLALIB, and accuracy compared to the United States Naval Observatory's NOVAS code.

We present results in this section, and provide the details of the programs in various appendices.

8.1 Internal Accuracy

We measured the internal accuracy of the TPM by transforming coordinates forward and back between every possible pair of states. We compute the angle between the original position vector and the one resulting from the forward and back trip, and plot the 21^2 errors in the histogram shown in Figure 5.

The TPM is reversible to within 4 milliarcseconds between any pair of states. We believe that the 4 mas errors result from simple least significant bit variations in the floating point calculations.

See Appendix C for details on the reversibility test program.

8.2 Accuracy Comparison with SLALIB

We compared the TPM to SLALIB for these calculations:

Calculation	flow
Galactic to FK4 Equatorial	S04-S05
FK4 to FK5 Equatorial	S05-S06
Ecliptic to FK5 Equatorial	S03-S02
Mean to Apparent	S06-S11
Apparent to Mean	S11-S06
Apparent to Observed	S11-S19

For each test, we drew 100,000 synthetic catalog positions from a 9-dimensional grid, using a uniformly distributed random number generator along each axis. The axes and their ranges are given as follows:

datum	name	minimum	maximum	units
ep0	catalog epoch	1950	2050	years
eq0	catalog eq.	1950	2050	years
ep1	current epoch	1950	2050	years
r0	α	0	360	degrees
d0	δ	-90	90	degrees
pr	$\dot{\alpha}$	-500	500	"/cy
pd	$\dot{\delta}$	-500	500	"/cy
px	parallax	0	1	"
rv	radial vel.	-100	100	km/s

Figure 6 shows error histograms for each of the tests. For the Apparent-to-Observed test, results were discarded if the resulting observed elevation was lower than 15°, so fewer errors are represented in the histogram for that test.

We see that the TPM matches the SLALIB results in all cases to within 5 milliarcseconds. As with the reversibility test, we attribute the residuals to LSB variations in the floating point calculations.

See Appendix D for more details.

8.3 Absolute Accuracy

We are not equipped to measure the absolute accuracy of the Telescope Pointing Machine directly with observations on the sky, so we measure "absolute accuracy" by comparing our results with those of a reliable third party. (We compare SLALIB to this reliable third party as well.) The United States Naval Observatory has kindly provided a list of apparent places of all FK5 stars for a specific epoch (UTC 2450136.3 = 1996 Feb 22) computed with the NOVAS (Naval Observatory Vector Astrometry Subroutines) code. This is the code used in producing the Astronomical Almanac, and can be considered properly calibrated.

To give a broader perspective on the results, we computed the USNO positions using both the TPM and SLALIB, which we already demonstrated to be consistent with each other. Figure 7 shows the error histograms giving the number of errors as a function of the error in milliarcseconds. The TPM histogram is shown with solid line, the SLALIB with a dashed line.

This figure shows that the TPM system closely matches SLALIB, with the two systems giving nearly indistinguishable error histograms. The broad peak of errors in the range of 2-10 milliarcseconds is due to the precision of the data provided by the U. S. Naval Observatory. The USNO listed the apparent place Right Ascensions to the nearest 1 millisecond of time, and Declinations to the nearest 10 milliarcseconds, so we cannot expect agreement on scales smaller than this.

8.4 Internal Timing

We timed various TPM calculations by executing each of them 10^4 times within a special test program, using the Unix `time(1)` command to measure the execution time. (See Appendix B for details about the timing test program.) We ran the test on three different computers: an SGI Indy with a 150 Mhz R4400 CPU running IRIX 5.3, a DECStation 5000/200 running Ultrix 4.4, and a Sparc 20 with a 60 Mhz CPU running Solaris. The results vary slightly with each run due to the multitasking nature of the workstations and the limitations of the Unix `time(1)` command, but they are probably accurate to better than 10%.

Table 3 shows the results, measured in milliseconds. The calculations shown include the Cartesian to spherical and spherical to Cartesian vector operations, the various sections of `tpm_data()`, each of the state machine transitions, and a number of "flows," each flow representing a typical pointing problem. See Figure 1 to decode the flows shown in Table 3.

8.5 Timing Comparison with SLALIB

We compared the performance of the TPM with SLALIB. This is a little tricky, because the programs modularize various calculations in different ways. Both systems separate the slow parts from the fast parts, and both allow the slow parts to be done less often. This effectively renders them irrelevant for serious timing comparisons. For example, Table 3 shows that the slow+refraction calculations take about 14 ms on an SGI Indy. If we assume that this needs to be done once per minute and we are in a 20 Hz telescope pointing loop, the 14 ms is averaged over 1200 loops for an effective per-loop load of 12 μs. This is clearly negligible. The real measure of performance is the part that must be done in every loop, and which cannot easily be broken down into any finer pieces.

For this timing test, we chose to time the transformation from mean to observed coordinates, and broke it up into four pieces: the setup for the mean-to-apparent flow (mappa in SLALIB parlance), the actual mean-to-apparent flow (mapqk), the setup for the apparent-to-observed flow (aoppa), and the actual apparent-to-observed flow (aoppat+aopqk). This last flow is the most important, because it must be done in every loop.

For comparison with the TPM, we chose a "best match" to the SLALIB flows: for the MAPPA part, we ran `tpm_data()` with the `fast`, `medium` and `slow` flags. For the MAPQK part, we ran `tpm()` from states S06 to S16. For the AOPPA part, we ran `tpm_data()` with the `fast` and `refraction` flags. Finally, for the AOPQK part we ran `tpm()` from states S16 to S19. Table 4 shows the results. For all practical purposes, there is no timing difference between TPM and SLALIB. SLALIB is a little faster on the infrequent MAPPA and MAPQK flows, a little slower on the infrequent AOPPA flow and the hard-real-time AOPQK flow. Both systems execute the time-critical flows (AOPQK) in well under a millisecond (on the SGI Indy R4400).

9 Subroutine Prototypes

9.1 Telescope Pointing Machine

The state data management routine is declared as follows:

```
void tpm_data(struct s_tstate *tstate, int flags)
```

where `tstate` points to the state data structure and `flags` is a bitfield specifying the desired calculations, whose bits are defined as follows:

flag name	action
TPM_INIT	initialize tstate with default values
TPM_FAST	do fast calculations
TPM_MEDIUM	do medium calculations
TPM_SLOW	do slow calculations
TPM_REFRACTION	do refraction calculations
TPM_ALL	do all calculations (except init)

The user can request a logical OR of any combination of them. If more than one set of calculations is requested, they will be performed in the proper order according to the dependencies shown in Fig 2.

The main TPM subroutine is declared as follows:

```
int tpm(struct s_v6 pvec[N_TPM_STATES],
int s1, int s2, double ep, double eq, struct s_tstate *tstate)
```

where `pvec` points to an array of state vectors, of which `pvec[s1]` is the desired starting state, `s2` is the desired end state, `ep` is the epoch of the state vector, `eq` is the equinox (if relevant) of the given state vector, and `tstate` points to the state data structure. `eq` is used only in transitions T01 and T02.

9.2 Proper Motion

Proper motion is not applied by the state machine. The user must apply proper motion to the state vector before invoking the state machine using the subroutine `proper_motion()`:

```
struct s_v6 proper_motion(struct s_v6 s, double t, double t0)
```

applies proper motion to the state vector $(\vec{p}, \vec{v})$ according to

$$\vec{p_2} = \vec{p_1} + \vec{v_1}(t - t_0) \tag{33}$$

Note that the state machine applies all its transformations to the velocity vector as well as the position vector, so the velocity vector is always correct for the given machine state. So, for example, equatorial velocities will be properly converted to galactic longitude and latitude rates of change.

9.3 Cartesian/Spherical Conversions

We provide routines for converting between Cartesian and spherical representations of state vectors. The TPM handles all its conversions in Cartesian coordinates, but conversions to and from spherical are needed to display results and read in catalog data.

The routine

```
struct s_v6 v6s2c(struct s_v6 v6)
```

converts from spherical coordinates to Cartesian according to

$$x = r \cos\delta \cos\alpha \tag{34}$$

$$y = r \cos\delta \sin\alpha \tag{35}$$

$$z = r \sin\delta \tag{36}$$

The velocity is computed by taking the first derivative of Equations 34-36:

$$\dot{x} = -r(\dot{\alpha} \cos\delta \sin\alpha + \dot{\delta} \sin\delta \cos\alpha) + \dot{r} \cos\delta \cos\alpha \tag{37}$$

$$\dot{y} = r(\dot{\alpha} \cos\delta \cos\alpha - \dot{\delta} \sin\delta \sin\alpha) + \dot{r} \cos\delta \sin\alpha \tag{38}$$

$$\dot{z} = r\dot{\delta} \cos\delta + \dot{r} \sin\delta \tag{39}$$

The routine

```
struct s_v6 v6c2s(struct s_v6 v6)
```

converts from Cartesian coordinates to spherical by inverting Equations 34-36:

$$r = \sqrt{x^2 + y^2 + z^2} \tag{40}$$

$$\alpha = \tan^{-1}(y/x) \tag{41}$$

$$\delta = \sin^{-1}(z/r) \tag{42}$$

The velocity is computed by taking the first derivative of Equations 40-42:

$$\dot{r} = \frac{x\dot{x} + y\dot{y} + z\dot{z}}{r} \tag{43}$$

$$\dot{\alpha} = \frac{x\dot{y} - y\dot{x}}{(r\cos\delta)^2} \tag{44}$$

$$\dot{\delta} = \frac{\dot{z} - \dot{r}\sin\delta}{r\cos\delta} \tag{45}$$

Note that in Equations 40-45 there can be singularities. We handle these explicitly, as follows.

1. If $r = 0$, then $\dot{r} = \dot{x}$ and $(\alpha, \delta, \dot{\alpha}, \dot{\delta}) = 0$
2. If $x = 0$, then $\alpha = (-\pi/2, 0, \pi/2)$ according to whether y is negative, zero, or positive.
3. Finally, if $\cos\delta = 0$, then $\dot{r} = \dot{z}/\sin\delta$ and either $\dot{\delta} = -\dot{y}/(r\sin\delta\sin\alpha)$ or $\dot{\delta} = -\dot{x}/(r\sin\delta\cos\alpha)$ depending on whether $\cos\alpha = 0$.

9.4 Accessing State Vector Data

The TPM libraries provide routines that allow the user to treat state vectors as "objects" whose internal structure is never seen by the programmer. State vectors are initialized with

```
struct s_v6 v6init(int type)
```

where `type` is either `CARTESIAN` or `SPHERICAL`.

The type can be queried with

```
int v6GetType(struct s_v6 v6),
```

but if you aren't sure of what type you have it's never wrong to enforce it:

```
v6 = v6s2c(v6)
```

will convert only if necessary; if `v6` is already in Cartesian form, no conversion will be attempted.

Cartesian components can be accessed using these calls:

```
double v6GetX(struct s_v6 v6);
double v6GetY(struct s_v6 v6);
double v6GetZ(struct s_v6 v6);
double v6GetXDot(struct s_v6 v6);
double v6GetYDot(struct s_v6 v6);
double v6GetZDot(struct s_v6 v6);
void v6SetX(struct s_v6 v6, double d);
void v6SetY(struct s_v6 v6, double d);
void v6SetZ(struct s_v6 v6, double d);
void v6SetXDot(struct s_v6 v6, double d);
void v6SetYDot(struct s_v6 v6, double d);
void v6SetZDot(struct s_v6 v6, double d);
```

Spherical components can be accessed using these calls:

```
double v6GetR(struct s_v6 v6);
double v6GetAlpha(struct s_v6 v6);
double v6GetDelta(struct s_v6 v6);
double v6GetRDot(struct s_v6 v6);
double v6GetAlphaDot(struct s_v6 v6);
double v6GetDeltaDot(struct s_v6 v6);
void v6SetR(struct s_v6 v6, double d);
void v6SetAlpha(struct s_v6 v6, double d);
void v6SetDelta(struct s_v6 v6, double d);
void v6SetRDot(struct s_v6 v6, double d);
void v6SetAlphaDot(struct s_v6 v6, double d);
void v6SetDeltaDot(struct s_v6 v6, double d);
```

9.5 Converting To and From Catalog Data

Star catalogs typically list entries in a spherical coordinate system, with parallax substituted for distance and mixed angular units of radians and arcseconds per century (the user must be careful to distinguish proper motions as arcseconds per *tropical* century or arcseconds per *Julian* century). Our state vectors, however, are represented as simple full-length 3-D positions and velocities whose units are (AU,AU/day) for Cartesian vectors and (AU,radians,AU/day,radians/day) for spherical vectors. We provide subroutines for going back and forth between catalog values and state vectors.

```
struct s_v6 cat2v6(
double r,       /* Right Ascension in radians */
double d,       /* Declination in radians */
double rd,      /* PM in RA (''/cy) */
double dd,      /* PM in Dec (''/cy) */
double px,      /* parallax in arcseconds */
double rv,      /* radial velocity in km/s */
double C)       /* number of days per century, tropical or Julian */
```

converts from catalog values to state vectors. For C, use either 36525 for Julian centuries or 36524.21987817305 for tropical centuries.

```
struct s_v6 v62cat(
double *r,      /* Right Ascension in radians */
double *d,      /* Declination in radians */
double *rd,     /* PM in RA (''/cy) */
double *dd,     /* PM in Dec (''/cy) */
double *px,     /* parallax in arcseconds */
double *rv,     /* radial velocity in km/s */
struct s_v6 v6; /* Cartesian state vector tobe converted */
double C)       /* number of days per century, tropical or Julian */
```

converts from state vectors to catalog values.

10 Examples

In the following examples, we show a few subroutines that act as "wrappers" around the TPM interface. Note that each varies only slightly from the others; regardless of the transformation being done, the TPM is invoked in the same way. The only real difference is in using different start and end states.

These subroutines are provided with the TPM distribution along with a main program that calls them, in the file `x_tpm_examples.c`. The main program is given here:

```
#include "astro.h"

int main(int argc, char *argv[])
{
    double lon, lat;
    double az, el;
    double r1, d1;
    double r2, d2;

    lon = 0.0;
    lat = d2r(90);
    tpm_gal(lon, lat, &r2, &d2);
    (void)fprintf(stdout, "tpm_gal (%s,%s)->(%s,%s)\n",
        fmt_alpha(lon),
        fmt_delta(lat),
        fmt_alpha(r2),
        fmt_delta(d2));

    r1 = 0.0;
    d1 = d2r(90);
    tpm_pre(r1, d1, &r2, &d2);
    (void)fprintf(stdout, "tpm_pre (%s,%s)->(%s,%s)\n",
        fmt_alpha(r1),
        fmt_delta(d1),
        fmt_alpha(r2),
        fmt_delta(d2));

    r1 = 0.0;
    d1 = d2r(90);
    tpm_mop(r1, d1, &az, &el);
    (void)fprintf(stdout, "tpm_mop (%s,%s)->(%s,%s)\n",
        fmt_alpha(r1),
        fmt_delta(d1),
        fmt_alpha(az),
        fmt_delta(el));

    return(0);
}
```

More extensive examples can be found in the SLALIB test directory `src/astro/sla_tpm`. See Appendix D for details.

10.1 Galactic to Equatorial

This example converts galactic coordinates (lon,lat) into FK4 B1950 equatorial coordinates.

```
/*************************/
/* galactic to FK4 B1950 */
/*************************/
static void
tpm_gal(
double lon,     /* galactic longitude */
double lat,     /* galactic latitude */
double *r,      /* RA */
double *d)      /* Dec */
{
    int s1;     /* start state */
    int s2;     /* end state */
    struct s_tstate tstate;
    struct s_v6 pvec[N_TPM_STATES];
    struct s_v6 v6;             /* state vector */

    /******************************/
    /* set up the target position */
    /******************************/
    v6 = v6init(SPHERICAL);
    v6SetR(v6, 1e9);
    v6SetAlpha(v6, lon);
    v6SetDelta(v6, lat);

    /*************************/
    /* don't need state data */
    /*************************/

    /******************/
    /* invoke the TPM */
    /******************/
    s1 = TPM_S04;
    s2 = TPM_S05;
    pvec[s1] = v6;
    (void)tpm(pvec, s1, s2, B1950, B1950, &tstate);
    v6 = pvec[s2];
    v6 = v6c2s(v6);     /* convert to spherical */
    *r = v6GetAlpha(v6);
    *d = v6GetDelta(v6);

    return;
}
```

10.2 Precession

This example precesses an FK5 J2000 position to date.

```
/**********************************/
/* precess from FK5 J2000 to date */
/**********************************/
static void
tpm_pre(
double r1,      /* RA */
double d1,      /* Dec */
double *r2,     /* RA */
double *d2)     /* Dec */
{
    int s1;     /* start state */
    int s2;     /* end state */
    struct s_tstate tstate;
    struct s_v6 pvec[N_TPM_STATES];
    struct s_v6 v6;             /* state vector */

    /******************************/
    /* set up the target position */
    /******************************/
    v6 = v6init(SPHERICAL);
    v6SetR(v6, 1e9);
    v6SetAlpha(v6, r1);
    v6SetDelta(v6, d1);

    /*************************/
    /* set up the state data */
    /*************************/
    tpm_data(&tstate, TPM_INIT);
    tstate.utc = utc_now();
    tpm_data(&tstate, TPM_ALL);

    /******************/
    /* invoke the TPM */
    /******************/
    s1 = TPM_S14;
    s2 = TPM_S15;
    pvec[s1] = v6;
    (void)tpm(pvec, s1, s2, J2000, J2000, &tstate);
    v6 = pvec[s2];
    v6 = v6c2s(v6);     /* convert to spherical */
    *r2 = v6GetAlpha(v6);
    *d2 = v6GetDelta(v6);

    return;
}
```

10.3 Mean to Observed

This example converts a mean FK5 J2000 position to topocentric observed azimuth and elevation.

```
/*****************************/
/* FK5 J2000 mean to observed */
/*****************************/
static void
tpm_mop(
double r1,      /* RA */
double d1,      /* Dec */
double *az,     /* azimuth */
double *el)     /* elevation */
{
    int s1;     /* start state */
    int s2;     /* end state */
    struct s_tstate tstate;
    struct s_v6 pvec[N_TPM_STATES];
    struct s_v6 v6;             /* state vector */

    /*****************************/
    /* set up the target position */
    /*****************************/
    v6 = v6init(SPHERICAL);
    v6SetR(v6, 1e9);
    v6SetAlpha(v6, r1);
    v6SetDelta(v6, d1);

    /************************/
    /* set up the state data */
    /************************/
    tpm_data(&tstate, TPM_INIT);
    tstate.utc = utc_now();
    /* choose a location in Wisconsin */
    tstate.lon = d2r(-90);
    tstate.lat = d2r(43);
    tpm_data(&tstate, TPM_ALL);

    /*****************/
    /* invoke the TPM */
    /*****************/
    s1 = TPM_S06;
    s2 = TPM_S19;
    pvec[s1] = v6;
    (void)tpm(pvec, s1, s2, J2000, J2000, &tstate);
    v6 = pvec[s2];
    v6 = v6c2s(v6);     /* convert to spherical */
    *az = v6GetAlpha(v6);
    *el = v6GetDelta(v6);

    return;
}
```

10.4 Tuning the Load

Table 3 shows that all the TPM transitions are quite fast, the slowest being T01 (precession to FK4 B1950) at about $340 \mu s$ on an SGI Indy. The expensive operations are in computing the state data, especially the slow state data (1.5 ms) and refraction model (12.5 ms). In a telescope control system, one needs to worry about spreading these calculations out over more pointing cycles.

Here is a code fragment from the WHAM Telescope Control Program, showing how the load is spread out in time (the tick variable increments once per second):

```
ptcs->tstate.utc = utc_now();
if (ptcs->tick % 3600 == 0) {
    tpm_data(&ptcs->tstate, TPM_SLOW|TPM_REFRACTION);
}
if (ptcs->tick % 60 == 0) {
    tpm_data(&ptcs->tstate, TPM_MEDIUM);
}
tpm_data(&ptcs->tstate, TPM_FAST);
```

Assuming we are doing an FK4-to-Observed flow (S01-S19 in Table 3) 20 times per second, the average time $\bar{T}$ consumed in each pointing cycle is

$$\bar{T} = 0.633 + 0.010 + \frac{0.085}{60 \times 20} + \frac{1.531 + 12.470}{3600 \times 20} \tag{46}$$

or about 0.643 ms, for a duty cycle of less than 2%. Note that spreading out the medium and slow `tpm_data()` calls over a minute and an hour, respectively, effectively eliminates them as a consideration.

In the WIYN control system, however, the *average* flow time was not the whole story, because the *peak* flow time was driven by the refraction call, which by itself took too much of the 50 ms loop time allowed by the TCS. We solved this simply by putting the slow and refraction calls to `tpm_data()` calls into a separate process, running continually at low priority. This used CPU cycles only as they were available, and still resulted in a refresh time of less than a minute, much faster than was necessary.

Another scheme to tune the load is suggested by Table 3. For the flow S01-S19, for example, one can transform the initial state vector from S01 to

S06 once, and then repeatedly transform the S06 state vector to S19. This saves about half the time per loop.

By the way, this is another subtle reason why using full 6-D state vectors, and transforming velocities along with positions, is very useful. If only positions were transformed, and if a target's space motion was so large that it needed to be applied continually (e.g. an asteroid), then S06 could not be used as an intermediate flow point because the space motion would still be expressed in the frame of S01. Because we transform the velocity to S06 as well, the space motion updates can be done in S06.

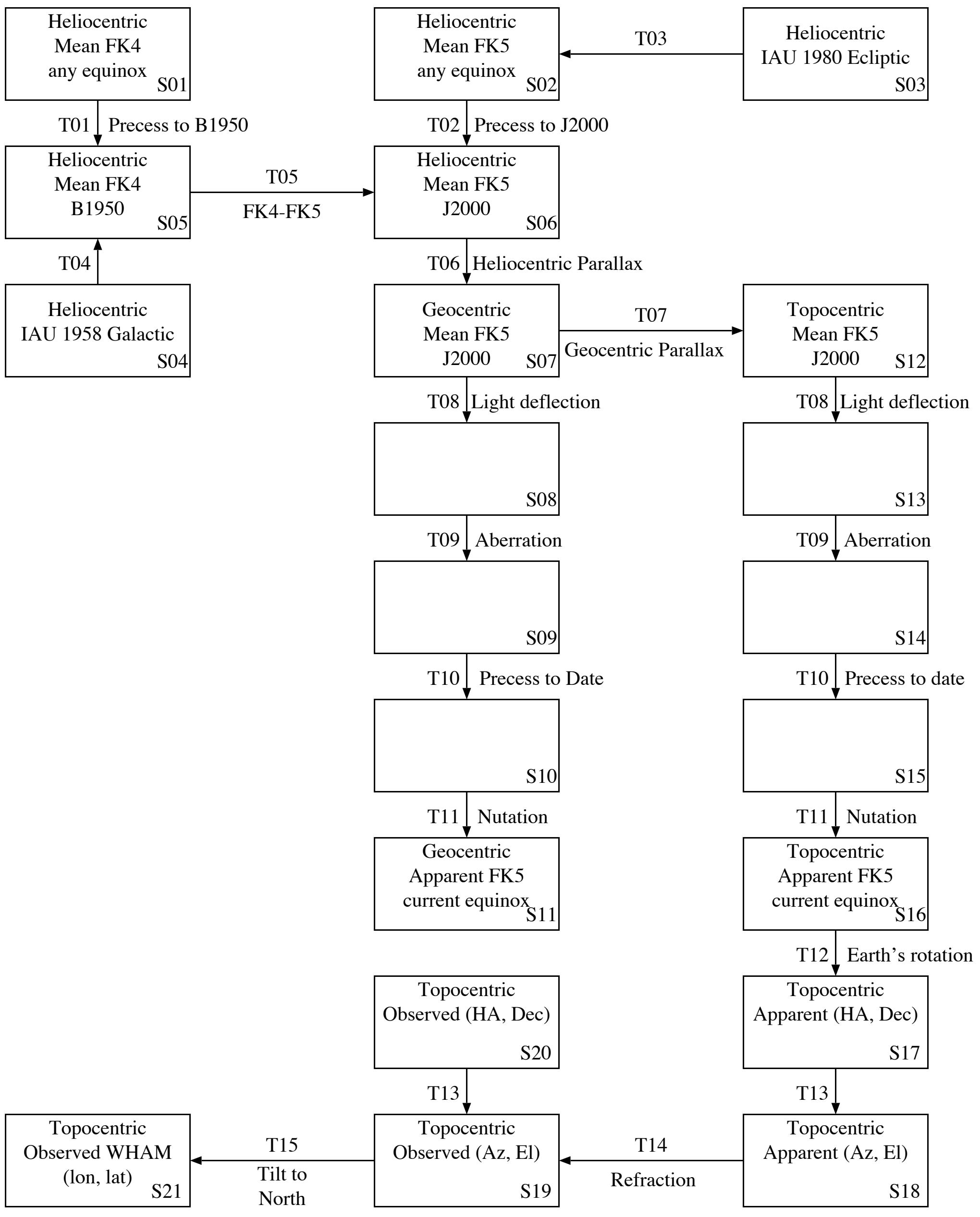
11 Summary

The Telescope Pointing Machine (TPM) is a fast, accurate, rigorous, compact, modern, and extendible implementation of high-precision astronomical pointing algorithms.

- Fast: it is as fast as the definitive procedural toolkit, SLALIB. The hard-real-time transformation of apparent to observed place occurs in 150 μs on an SGI Indy with a 150 Mhz R4400 CPU.
- Accurate: it compares to SLALIB to within 5 milliarcseconds, and with data tabulated by the United Stated Naval Observatory to within the 10 milliarcsecond precision of their provided data.
- Rigorous: No corners are cut in algorithms. For example, only sub-milliarcsecond effects are ignored when computing the gravitational aberration of light, and the effect of space motion on parallax and radial velocity is included.
- Compact: The TPM system runs in the WIYN 3.5m Telescope Control System, and compiles to an object file size of less than 60 kilobytes.
- Modern: The TPM system is a newly written set of ANSI-C subroutines with no vendor or platform specific features.
  - Orthogonal: The software is highly modularized, with no operational penalty, such that no algorithm (e.g. the value of the obliquity of the ecliptic) exists in more than one place.
  - Vector/matrix formulation: we use a vector and matrix formulation throughout. We use no spherical trigonometry or low-order trigonometric expansions in the state transitions.
- Extendible: by adding states and updating the state table, users can fully integrate new reference frames into the TPM and effectively connect them to all other states.

12 Acknowledgements

We wish to thank Michael Nastvogel-Wörz of the European Southern Observatory for performing a timing and accuracy comparison with SLALIB that pointed out areas for improving the Telescope Pointing Machine, We wish to thank Dr. Thomas E. Corbin and Mr. Ellis Holdenried of the United States Naval Observatory's Astrometry Department for providing the definitive list of FK5 apparent places. Finally, we wish to thank Dr. Patrick Wallace of the Rutherford Appleton Laboratory for the generous use of his time and expertise in discussing many points, both grand and obscure, that are relevant to high precision astrometric calculations.



Heliocentric Mean FK4 any equinox S01
Heliocentric Mean FK5 any equinox S02
T03
Heliocentric IAU 1980 Ecliptic S03
T01 Precess to B1950
T02 Precess to J2000
Heliocentric Mean FK4 B1950 S05
T05 FK4-FK5
Heliocentric Mean FK5 J2000 S06
T04
T06 Heliocentric Parallax
Heliocentric IAU 1958 Galactic S04
Geocentric Mean FK5 J2000 S07
T07 Geocentric Parallax
Topocentric Mean FK5 J2000 S12
T08 Light deflection
T08 Light deflection
S08
S13
T09 Aberration
T09 Aberration
S09
S14
T10 Precess to Date
T10 Precess to date
S10
S15
T11 Nutation
T11 Nutation
Geocentric Apparent FK5 current equinox S11
Topocentric Apparent FK5 current equinox S16
T12 Earth's rotation
Topocentric Observed (HA, Dec) S20
Topocentric Apparent (HA, Dec) S17
T13
T13
Topocentric Observed WHAM (lon, lat) S21
T15 Tilt to North
Topocentric Observed (Az, El) S19
T14 Refraction
Topocentric Apparent (Az, El) S18

Figure 1: Telescope Pointing Machine State Diagram

Telescope Pointing Machine
Version $Revision: 1.12 $ ($Date: 2002/05/22 11:50:06 $)

(based on the Keck pointing flow by P. T. Wallace)

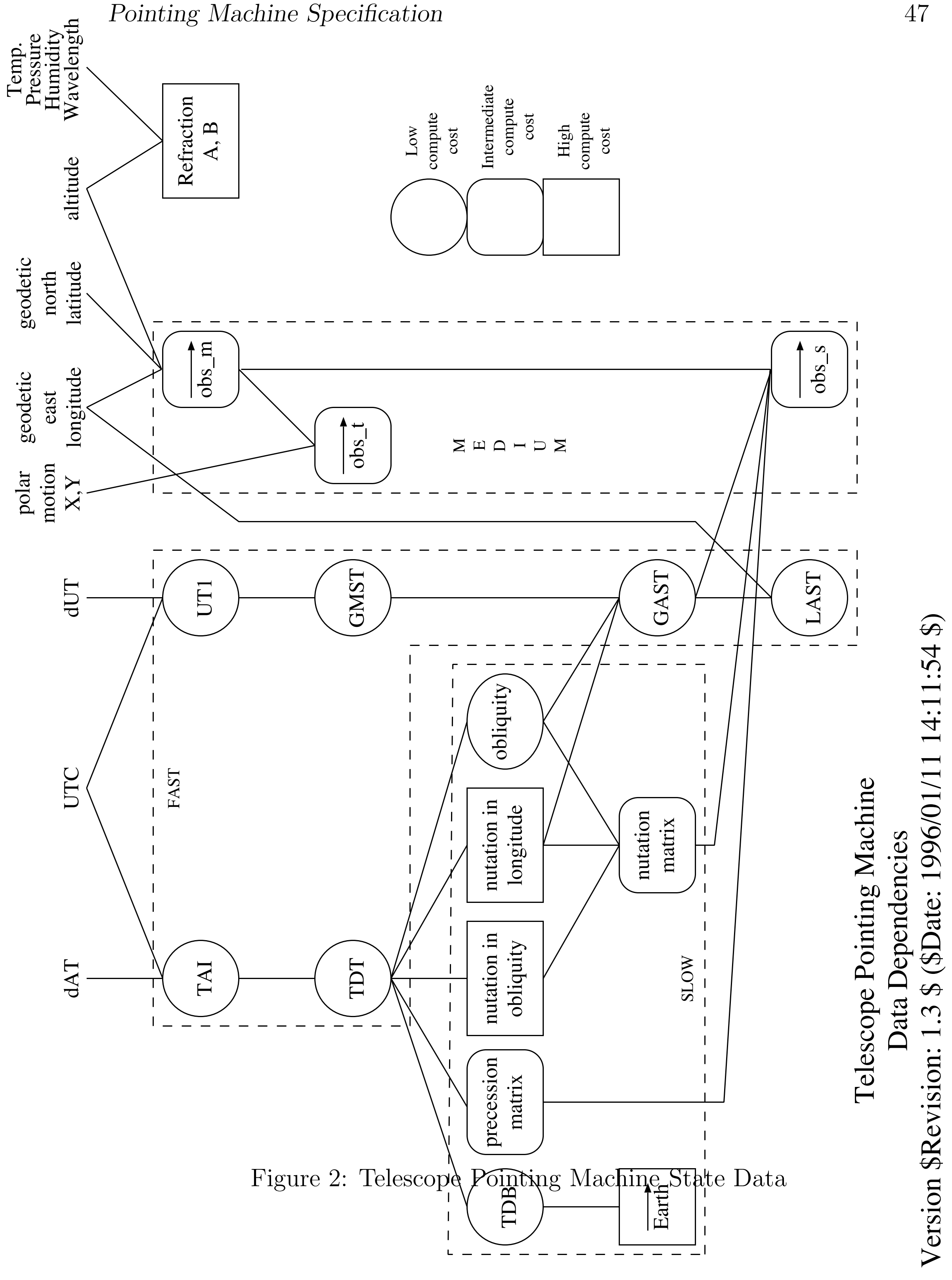
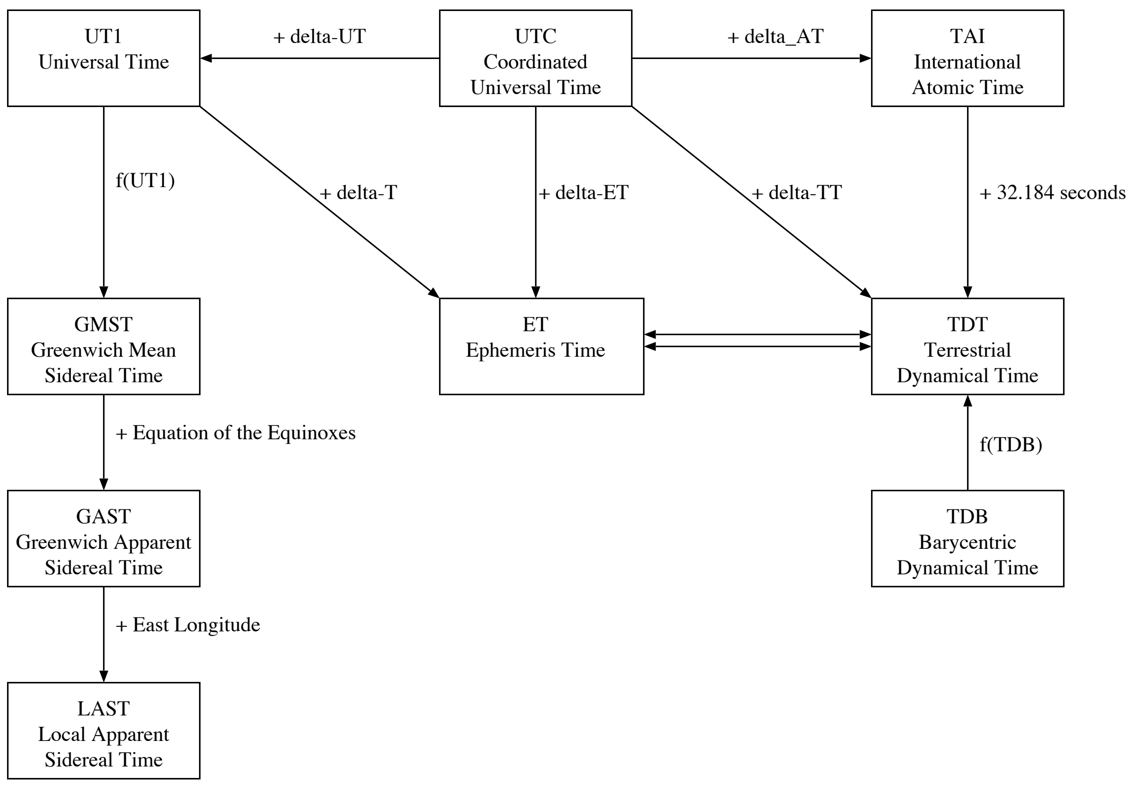


Figure 2: Telescope Pointing Machine State Data



Telescope Pointing Machine
Time Transitions
Version $Revision: 1.4 $ ($Date: 1996/02/06 11:36:53 $)

Figure 3: Telescope Pointing Machine Time Transitions

```
struct s_tstate {
        /*************************/
        /* independent variables */
        /*************************/
        double utc;             /* coordinated universal time, in JD */
        int delta_at;           /* utc + delta_at = tai */
        double delta_ut;        /* utc + delta_ut = ut1 */
        double lon;             /* east longitude in radians */
        double lat;             /* latitude in radians */
        double alt;             /* altitude above geoid in meters */
        double xpole;           /* polar motion in radians */
        double ypole;           /* polar motion in radians */
        double T;               /* ambient temperature in Kelvins */
        double P;               /* ambient pressure in millibars */
        double H;               /* ambient humidity (0-1) */
        double wavelength;      /* observing wavelength in microns */

        /*****************************/
        /* dependent dynamical times */
        /*****************************/
        double tai;             /* international atomic time */
        double tdt;             /* terrestrial dynamical time */
        double tdb;             /* barycentric dynamical time */

        /************************************/
        /* dependent geometrical quantities */
        /************************************/
        double obliquity;       /* the obliquity of the ecliptic */
        double nut_lon;         /* the nutation in longitude */
        double nut_obl;         /* the nutation in the obliquity */
        struct s_m3 nm;         /* the nutation matrix for now */
        struct s_m3 pm;         /* the precession matrix from J2000 to now */

        /******************************/
        /* dependent rotational times */
        /******************************/
        double ut1;             /* universal time */
        double gmst;            /* greenwich mean sidereal time */
        double gast;            /* greenwich apparent sidereal time */
        double last;            /* local apparent sidereal time */

        /************************/
        /* observer ephemerides */
        /************************/
        struct s_v6 eb;         /* barycentric earth state vector */
        struct s_v6 eh;         /* heliocentric earth state vector */
        struct s_v6 obs_m;      /* geocentric earth-fixed mean state */
        struct s_v6 obs_t;      /* geocentric earth-fixed true state */
        struct s_v6 obs_s;      /* geocentric space-fixed mean state */

        /*********************************/
        /* dependent physical quantities */
        /*********************************/
        double refa;            /* refraction coefficient */
        double refb;            /* refraction coefficient */
};
```

Figure 4: Telescope Pointing Machine State Data Structure

Reversibility errors in milliarcseconds

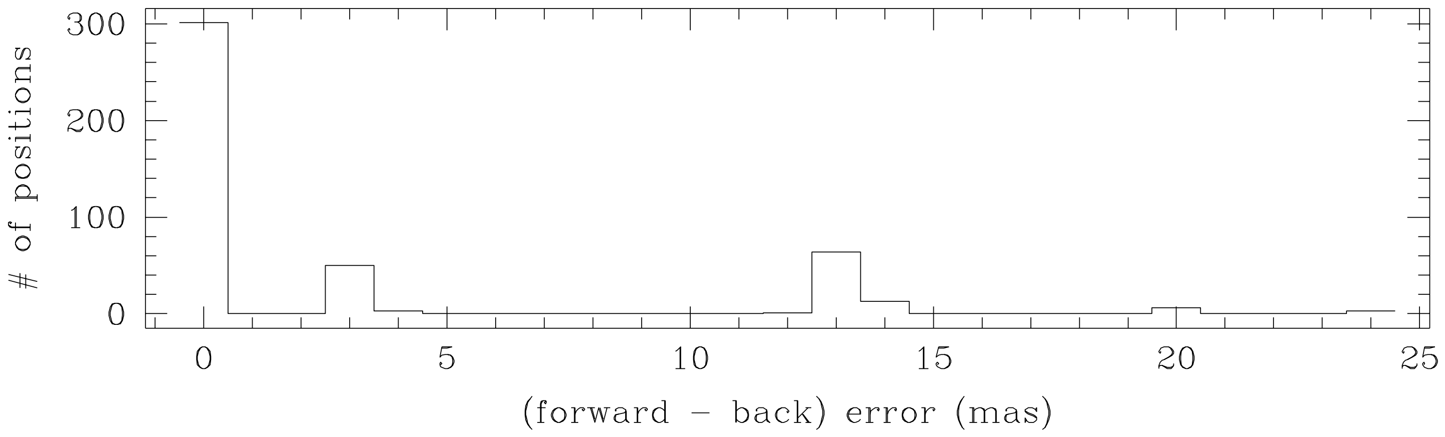


Figure 5: State Machine Reversibility Errors

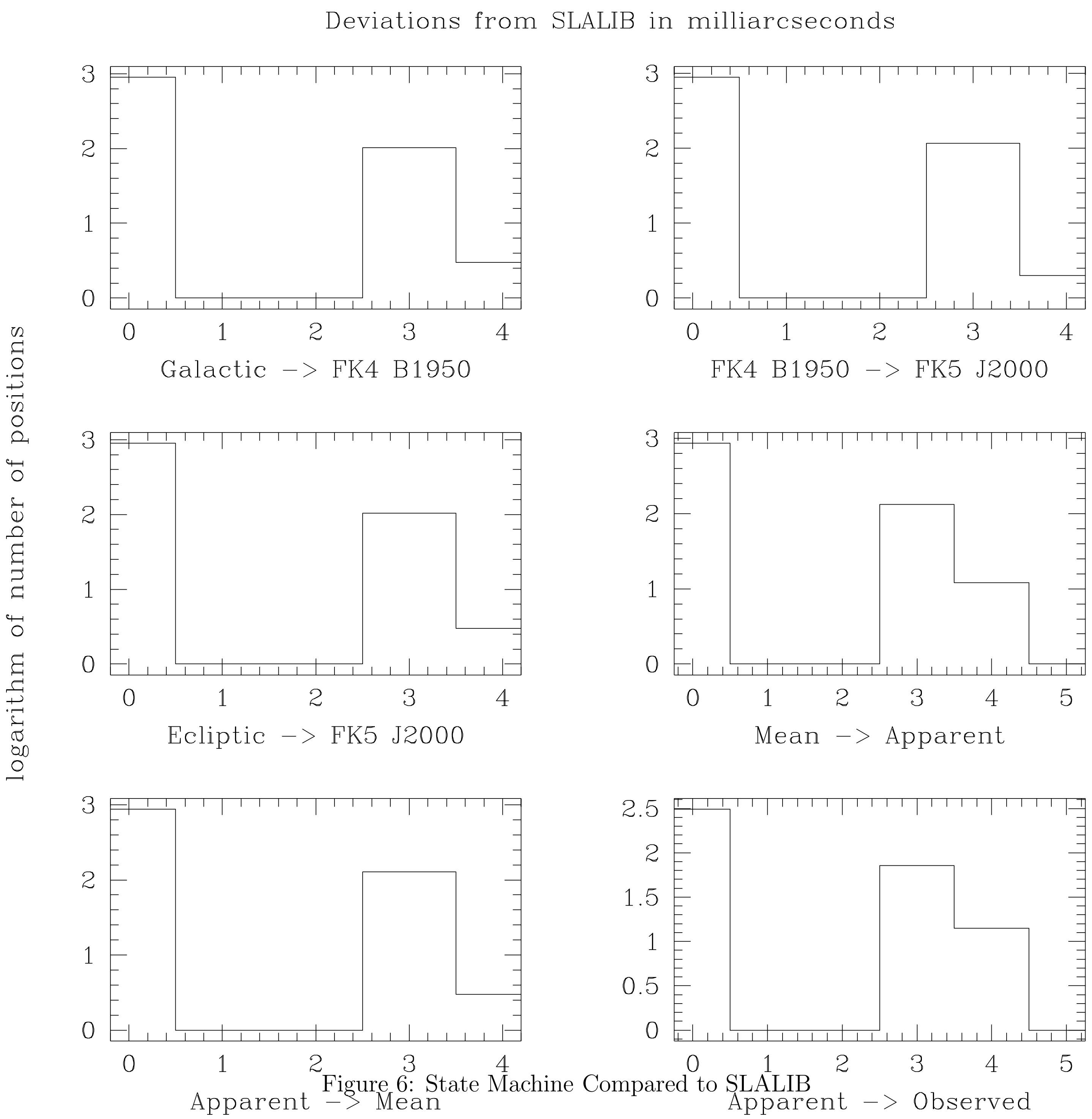


Figure 6: State Machine Compared to SLALIB

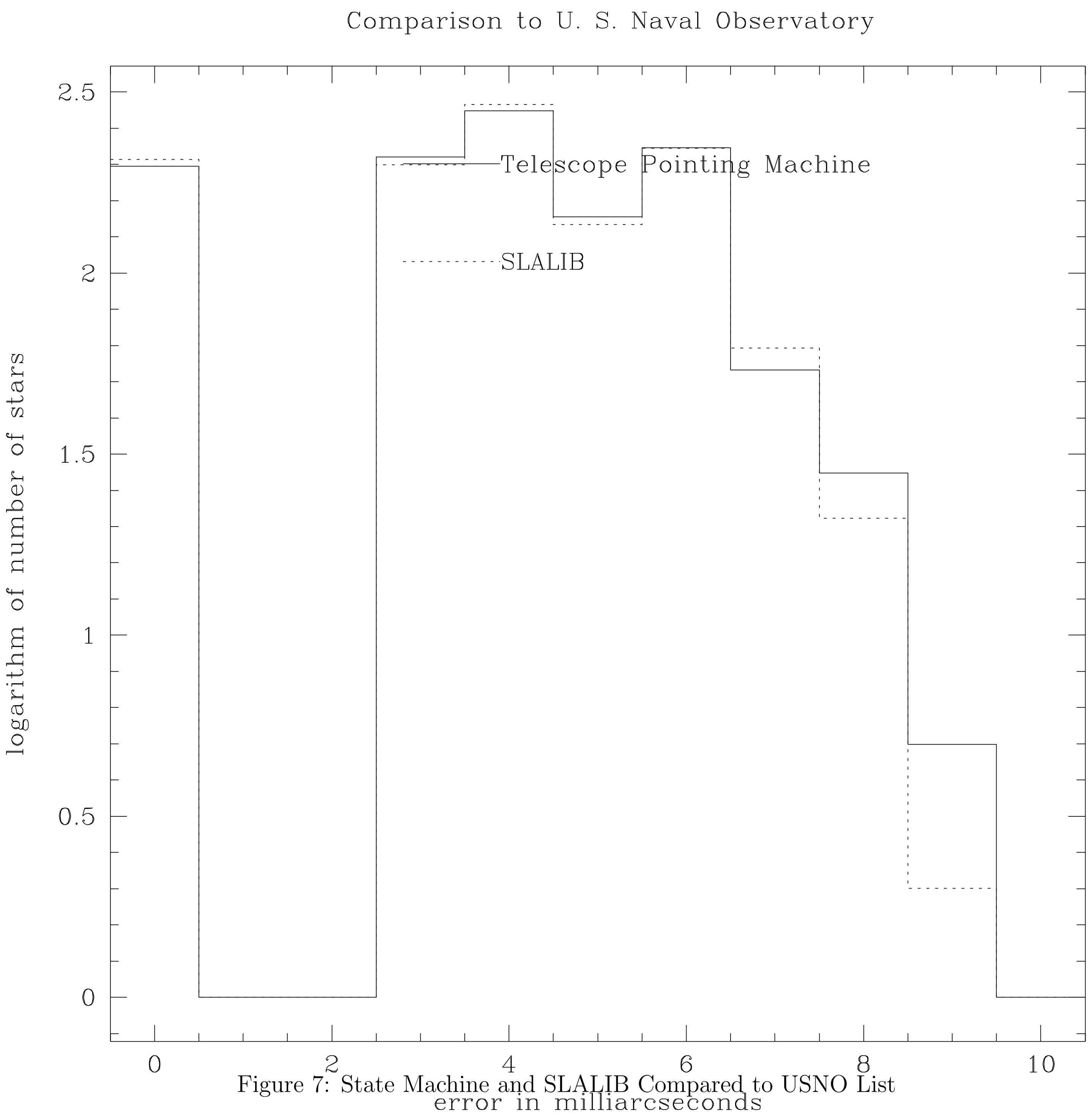


Figure 7: State Machine and SLALIB Compared to USNO List

	destination state									
state	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10
S01	+T00	+T01	+T01	+T01	+T01	+T01	+T01	+T01	+T01	+T01
	S01	S05	S05	S05	S05	S05	S05	S05	S05	S05
S02	+T02	+T00	-T03	+T02	+T02	+T02	+T02	+T02	+T02	+T02
	S06	S02	S03	S06	S06	S06	S06	S06	S06	S06
S03	+T03	+T03	+T00	+T03	+T03	+T03	+T03	+T03	+T03	+T03
	S02	S02	S03	S02	S02	S02	S02	S02	S02	S02
S04	+T04	+T04	+T04	+T00	+T04	+T04	+T04	+T04	+T04	+T04
	S05	S05	S05	S04	S05	S05	S05	S05	S05	S05
S05	-T01	+T05	+T05	-T04	+T00	+T05	+T05	+T05	+T05	+T05
	S01	S06	S06	S04	S05	S06	S06	S06	S06	S06
S06	-T05	-T02	-T02	-T05	-T05	+T00	+T06	+T06	+T06	+T06
	S05	S02	S02	S05	S05	S06	S07	S07	S07	S07
S07	-T06	-T06	-T06	-T06	-T06	-T06	+T00	+T08	+T08	+T08
	S06	S06	S06	S06	S06	S06	S07	S08	S08	S08
S08	-T08	-T08	-T08	-T08	-T08	-T08	-T08	+T00	+T09	+T09
	S07	S07	S07	S07	S07	S07	S07	S08	S09	S09
S09	-T09	-T09	-T09	-T09	-T09	-T09	-T09	-T09	+T00	+T10
	S08	S08	S08	S08	S08	S08	S08	S08	S09	S10
S10	-T10	-T10	-T10	-T10	-T10	-T10	-T10	-T10	-T10	+T00
	S09	S09	S09	S09	S09	S09	S09	S09	S09	S10
S11	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11
	S10	S10	S10	S10	S10	S10	S10	S10	S10	S10
S12	-T07	-T07	-T07	-T07	-T07	-T07	-T07	-T07	-T07	-T07
	S07	S07	S07	S07	S07	S07	S07	S07	S07	S07
S13	-T08	-T08	-T08	-T08	-T08	-T08	-T08	-T08	-T08	-T08
	S12	S12	S12	S12	S12	S12	S12	S12	S12	S12
S14	-T09	-T09	-T09	-T09	-T09	-T09	-T09	-T09	-T09	-T09
	S13	S13	S13	S13	S13	S13	S13	S13	S13	S13
S15	-T10	-T10	-T10	-T10	-T10	-T10	-T10	-T10	-T10	-T10
	S14	S14	S14	S14	S14	S14	S14	S14	S14	S14
S16	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11	-T11
	S15	S15	S15	S15	S15	S15	S15	S15	S15	S15
S17	-T12	-T12	-T12	-T12	-T12	-T12	-T12	-T12	-T12	-T12
	S16	S16	S16	S16	S16	S16	S16	S16	S16	S16
S18	-T13	-T13	-T13	-T13	-T13	-T13	-T13	-T13	-T13	-T13
	S17	S17	S17	S17	S17	S17	S17	S17	S17	S17
S19	-T14	-T14	-T14	-T14	-T14	-T14	-T14	-T14	-T14	-T14
	S18	S18	S18	S18	S18	S18	S18	S18	S18	S18
S20	+T13	+T13	+T13	+T13	+T13	+T13	+T13	+T13	+T13	+T13
	S19	S19	S19	S19	S19	S19	S19	S19	S19	S19
S21	-T15	-T15	-T15	-T15	-T15	-T15	-T15	-T15	-T15	-T15
	S19	S19	S19	S19	S19	S19	S19	S19	S19	S19

Table 1: State Table (part 1) for the Telescope Pointing Machine.

	destination state										
state	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
S01	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05	+T01 S05
S02	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06	+T02 S06
S03	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02	+T03 S02
S04	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05	+T04 S05
S05	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06	+T05 S06
S06	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07	+T06 S07
S07	+T08 S08	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12	+T07 S12
S08	+T09 S09	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07	-T08 S07
S09	+T10 S10	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08	-T09 S08
S10	+T11 S11	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09	-T10 S09
S11	+T00 S11	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10	-T11 S10
S12	-T07 S07	+T00 S12	+T08 S13	+T08 S13	+T08 S13	+T08 S13	+T08 S13	+T08 S13	+T08 S13	+T08 S13	+T08 S13
S13	-T08 S12	-T08 S12	+T00 S13	+T09 S14	+T09 S14	+T09 S14	+T09 S14	+T09 S14	+T09 S14	+T09 S14	+T90 S14
S14	-T09 S13	-T09 S13	-T09 S13	+T00 S14	+T10 S15	+T10 S15	+T10 S15	+T10 S15	+T10 S15	+T10 S15	+T10 S15
S15	-T10 S14	-T10 S14	-T10 S14	-T10 S14	+T00 S15	+T11 S16	+T11 S16	+T11 S16	+T11 S16	+T11 S16	+T11 S16
S16	-T11 S15	-T11 S15	-T11 S15	-T11 S15	-T11 S15	+T00 S16	+T12 S17	+T12 S17	+T12 S17	+T12 S17	+T12 S17
S17	-T12 S16	-T12 S16	-T12 S16	-T12 S16	-T12 S16	-T12 S16	+T00 S17	+T13 S18	+T13 S18	+T13 S18	+T13 S18
S18	-T13 S17	-T13 S17	-T13 S17	-T13 S17	-T13 S17	-T13 S17	-T13 S17	+T00 S18	+T14 S19	+T14 S19	+T14 S19
S19	-T14 S18	-T14 S18	-T14 S18	-T14 S18	-T14 S18	-T14 S18	-T14 S18	-T14 S18	+T00 S19	-T13 S20	+T15 S21
S20	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T13 S19	+T00 S20	+T13 S19
S21	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	-T15 S19	+T00 S21

Table 2: State Table (part 2) for the Telescope Pointing Machine.

transition	execution time (ms)		
	Indy R4400 (150 MHz)	DECStation 5000/200	Sparc 20 (60 MHz)
c2s	0.017	0.050	0.020
s2c	0.009	0.020	0.010
init	0.023	0.090	0.030
slow	1.531	6.180	2.540
medium	0.085	0.360	0.100
fast	0.010	0.030	0.010
refraction	12.470	46.280	18.800
T01	0.334	1.380	0.340
T02	0.256	1.170	0.260
T03	0.021	0.070	0.020
T04	0.078	0.280	0.080
T05	0.061	0.190	0.040
T06	0.009	0.020	< 0.010
T07	0.008	0.020	< 0.010
T08	0.035	0.100	0.020
T09	0.015	0.050	0.010
T10	0.024	0.100	0.020
T11	0.010	0.040	0.010
T12	0.051	0.200	0.050
T13	0.033	0.120	0.040
T14	0.042	0.150	0.050
T15	0.032	0.120	0.040
S01-S05	0.333	1.390	0.340
S01-S06	0.402	1.580	0.380
S01-S07	0.407	1.580	0.390
S01-S12	0.411	1.620	0.390
S01-S13	0.467	1.780	0.410
S01-S14	0.468	1.840	0.430
S01-S15	0.489	1.970	0.440
S01-S16	0.502	1.990	0.450
S01-S17	0.559	2.140	0.500
S01-S18	0.585	2.320	0.540
S01-S19	0.633	2.310	0.620

Table 3: Execution Times for State Transitions

	execution time (ms)					
	Indy R4400		DS5000/200		Sparc 20	
calculation	TPM	SLALIB	TPM	SLALIB	TPM	SLALIB
mappa	1.83	0.72	7.06	2.65	2.68	1.50
mapqk	0.31	0.03	2.63	0.12	0.05	0.05
aoppa	12.54	16.17	46.69	67.71	19.26	26.34
aopqk	0.15	0.71	0.46	3.84	0.16	1.21

Table 4: TPM/SLALIB Timing Comparisons

A FK4/FK5 Test Programs

We were very careful in implementing this transformation, because our state vectors differ in units from those of the ES and Yallop. In the literature, the position vector is a unit vector and the velocity vector, while Cartesian in nature, has mixed units of km/s and arcseconds per century. The implicit units conversions are built into the 6x6 matrices given in Yallop and the ES. In our implementation, the state vectors are true 3-D positions and velocities whose units are simply AU and AU/day. Therefore, parts of our 6x6 matrix are scaled differently.

We approached this as follows. First, we made sure we could recreate the standard matrix. The program `x_fk45.c` does this, from "first principles", following the recipe in Yallop. After creating the matrix, is then reads in the file `fk45.dat`, which represents the FK4 catalog entries given in Table 3.58.1 of the ES. The program prints out the calculated FK5 entries, matching those of Table 3.58.1. Note that command line options can force this program to use the precession angles of Andoyer (as in Yallop) or Kinoshita (as in the ES).

Second, we calculated our new, modified matrix, again according to Yallop, but without the various unnecessary units conversions. The program `x_fk45x.c` calculates the modified matrix, and then as before performs the conversions in the ES Table 3.58.1.

The matrix generated by `x_fk45x.c` is the one actually coded into the TPM's subroutine library. That is, the output of `x_fk45x.c` was edited and transcribed into the subroutine `fk425()`.

Finally, we needed to test that the transcription mentioned above was done correctly. The program `x_fk45xx.c` also performs the conversions in the ES Table 3.58.1, but it does so using the TPM's library routine `fk425()` instead of having calculated the matrix itself. Again, the FK5 output matches the answers in the ES Table 3.58.1.

These three test programs have their mirror-image counterparts in `x_fk54.c`, `x_fk54x.c`, `x_fk54xx.c`, and `k54.dat`.

B State Machine Timing Program

The TPM timing program is called `x_tpm_timer.c`. Command line options to `x_tpm_timer.c` control the number of executions as well as the particular calculation that is to be timed. Run the program with the `help` option to get a short summary of options.

Here is the shell script `x_tpm_timer.sh` used to produce Table 3.

```
#! /bin/sh

pgm="x_tpm_timer.$MACHINE"
log="$pgm.out"

N="10000"

cp /dev/null $log

/bin/time $pgm -v -n $N -c2s 2>> $log
/bin/time $pgm -v -n $N -s2c 2>> $log

/bin/time $pgm -v -n $N -init 2>> $log
/bin/time $pgm -v -n $N -slow 2>> $log
/bin/time $pgm -v -n $N -medium 2>> $log
/bin/time $pgm -v -n $N -fast 2>> $log
/bin/time $pgm -v -n $N -refraction 2>> $log

for trans in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    /bin/time $pgm -v -n $N -trans $trans 2>> $log
done

s1=1
for s2 in 5 6 7 12 13 14 15 16 17 18 19
do
    /bin/time $pgm -v -n $N -flow $s1 $s2 2>> $log
done
```

C State Machine Reversibility Program

The state machine reversibility program is called `tpm_fwd_bck.c`. Command line options to `x_tpm_timer.c` control the number of executions as well as the particular calculation that is to be timed. Run the program with the `help` option to get a short summary of options.

Here is the shell script `x_tpm.sh` used to produce Figure 5.

```
#! /bin/sh
# use x_tpm to make a histogram of fwd/bck errors

pgm="x_tpm.$MACHINE"

# program to make a histogram
HIST="../streams/hist.$MACHINE -nbins 10000 -ymin 0 -ymax 10000"

# program to take the log of the y values
WINDOW="../streams/window.$MACHINE -ylog10"

$pgm | grep err | awk '{print NR, $NF}' | $HIST > $pgm.hist
$WINDOW < $pgm.hist > $pgm.hist.lg
```

D SLALIB Test Suite

SLALIB is considered to be definitive in the area of pointing telescopes, so a SLALIB comparison is required. (See the Introduction for ways in which the Telescope Pointing Machine differs from SLALIB.)

The SLALIB comparison programs are in the `src/astro/sla_tpm` subdirectory of the TPM distribution.

We first generated a synthetic stellar catalog with the program `sla_tpm_data.c`. This program draws catalog entries from a 9-dimensional grid of

$$(\alpha, \delta, eq, \dot{\alpha}, \dot{\delta}, ep, \pi, \dot{r}, utc)$$

where *eq* is the catalog equinox, *ep* is the epoch of proper motions, and *utc* is the time for which a position is to be calculated. For each axis of the grid, a value is chosen at random, with uniform probability, from the ranges shown in Section 8.2. The user can select the number of catalog entries generated; we chose 10^5.

A separate program makes each of the comparison calculations:

Calculation	flow	program name
Galactic to FK4 Equatorial	S04-S05	`sla_tpm_gal.c`
FK4 to FK5 Equatorial	S05-S06	`sla_tpm_fk45.c`
Ecliptic to FK5 Equatorial	S03-S02	`sla_tpm_ecl.c`
Mean to Apparent	S06-S11	`sla_tpm_map.c`
Apparent to Mean	S11-S06	`sla_tpm_amp.c`
Apparent to Observed	S11-S19	`sla_tpm_aop.c`
Timing comparison	various	`sla_tpm_timer.c`

See `sla_tpm.sh` and `sla_tpm_timer.sh` for examples on running these programs.

Using the `verbose` option causes the programs to print out intermediate results. Without this option, you just get one line per input catalog entry, giving as the last item on the line the deviation from SLALIB in milliarcseconds.