SolTrack: a free, fast and accurate routine to compute the position of the Sun

Marc van der Sluys[a,b,*], Paul van Kan[c]

[a]*Nikhef, P.O. Box 41882, NL-1009 DB Amsterdam, The Netherlands.*
[b]*Utrecht University, Institute for Gravitational and Subatomic Physics (GRASP), P.O. Box 80000, NL-3508 TA Utrecht, The Netherlands.*
[c]*HAN University of Applied Sciences, P.O. Box 2217, NL-6802 CE, Arnhem, The Netherlands.*

Abstract

We present a simple, free, fast and accurate C/C++ and Python routine called *SolTrack*, which can compute the position of the Sun at any instant and any location on Earth. The code allows tracking of the Sun using a low-specs embedded processor, such as a PLC or a microcontroller, and can be used for applications in the field of (highly) concentrated (photovoltaic) solar power ((H)CPV and CSP), such as tracking control and yield modelling. *SolTrack* is accurate, fast and open in its use, and compares favourably with similar algorithms that are currently available for solar tracking and modelling. *SolTrack* computes 1.5×10^6 positions per second on a single 2.67 GHz CPU core. For the period between the years 2017 and 2116 the uncertainty in position is $0.0036 \pm 0.0042°$, that in solar distance $0.0017 \pm 0.0029\%$. In addition, *SolTrack* computes rise, transit and set times to an accuracy better than 1 second. The code is freely available online.[1]

Keywords: Solar tracking, algorithm, CPV, CSP

1. Introduction

For a project to develop affordable units for highly-concentrated photovoltaics (HCPV) in an urban environment, we require an accurate ($\lesssim 0.01°$), fast and affordable algorithm that we can use freely for commercial products. Currently, existing codes in the field are either less accurate or have an insufficiently open licence. Since many codes that could potentially fulfil our demands can be found online, albeit not currently in the field of solar concentration or in a preferred language (C/C++ and Python), we considered a number of these codes and used them to develop the *SolTrack* code. Since many steps in any algorithm to compute the position of the Sun can be performed in a number of ways, we selected the alternatives that gave the speed-accuracy performance of our liking and assembled them to create *SolTrack*. We currently use our code to control our HCPV system and to model its electrical yield. In Section 2 we describe the origin and key features of the code that is included in *SolTrack* and the software licence under which it can be used. In Section 3 we present the accuracy and computational speed of our code, and compare them to the two other algorithms that are currently available in the field. In Section 4 we summarise our findings and present our conclusions.

2. Description of the *SolTrack* code

The original *SolTrack* code (van der Sluys and van Kan, 2014–2022) is written in plain C, which makes it available for many users and computing platforms. In addition, there is a Python version available. It can compute the position of the Sun in topocentric coordinates, for any geographical location on Earth and for any instant. The code can express the solar vector both in a horizontal (azimuth and altitude or zenith angle) and in a parallactic coordinate system (using the equatorial coordinates hour angle and declination). The *SolTrack* code is derived from the Fortran library *libTheSky* (van der Sluys, 2002–2022) and includes corrections for aberration and parallax, and a simple routine to correct for atmospheric refraction (Saemundsson, 1986). Apart from sky position and distance, *SolTrack* can compute rise, set

*Corresponding author
Email address: `sluys@nikhef.nl` (Marc van der Sluys)
[1]http://soltrack.sf.net, https://pypi.org/project/soltrack/

and transit times for a given location and day, as well as the corresponding azimuths and altitude. The C/C++ code is freely available online at `http://soltrack.sf.net`, under the terms and conditions of the GNU Lesser General Public Licence (Free Software Foundation, 2007). The Python code can be freely downloaded from the Python Package Index (PyPI) at `https://pypi.org/project/soltrack/`. A few examples that use *SolTrack* are added to the download, so that the users can verify their implementation.

The high accuracy, low computational cost and implementation in plain C and Python allow a flexible use of *SolTrack*, ensuring that the code can run on inexpensive, low-spec embedded systems like the STM32F4 DISCOVERY boards, on simple systems with light-weight operating systems like the Raspberry Pi, on PLCs and on standard PCs or servers. A version ported to Arduino Scratch is available on the website. In addition to *SolTrack*, we have developed a closed-loop system that allows feedback from sensors near the PV cell in order to correct for discrepancies that arise due to *e.g.* an imperfect installation of the system, or mechanical deformations. Furthermore, we have designed an algorithm that takes into account non-perfect alignment, for example because the system is aligned with the main axes of the building it resides in, rather than the north-south axis (van der Sluys et al., 2015).

2.1. Calculation of the position of the Sun

The open-source Fortran library *libTheSky* (van der Sluys, 2002–2022) contains several routines to compute the Sun's position with different performances in terms of accuracy and computational speed. We used the subroutine `sunpos_la()` (for a low-accuracy Sun position[2]) as a development environment. We used the references in the code to read the literature behind it and see whether alternatives existed for each equation. Thus we were able to test the influence of the different parts of the code and their alternatives on the accuracy and speed of the code. In many cases, we simplified the original equations by leaving out higher-order terms if the resulting increase in speed was significant and the loss in accuracy was limited or negligible. In some cases we found alternatives with a similar effect. We translated the resulting code from Fortran to C and Python as the basis for *SolTrack*, in order to meet our own demands, as well as to make it available to a large audience and a wide range of computing platforms.

By *profiling* our code using gprof (part of GNU binutils; The GNU Project, 2022) and Valgrind (The Valgrind Developers, 2022) we found that the majority of the CPU cycles went to computing sines, cosines and tangents and to the use of the `fmod()` function. We reduced CPU time by computing the trigonometric functions once and distributing the result through the code wherever that was possible. Some angles, like the latitude, declination and altitude, are defined in the range of $[-\frac{\pi}{2}, \frac{\pi}{2}]$ ($[-90°, 90°]$) so that the cosine of an angle φ can be computed cheaply from $\cos\varphi = \sqrt{1-\sin^2\varphi}$ without ending up in the wrong quadrant. If all three of sine, cosine and tangent of an angle must be computed, the latter can be cheaply obtained by division of the first two. In addition, we wrote our own `atan2()` function in C, which computes the inverse tangent without confusion in the quadrants, based on Wikipedia (2019). If $\tan\alpha \equiv \frac{y}{x}$, then

$$\alpha \equiv \mathrm{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0; \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0; \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0; \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0; \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0; \\ 0 & \text{if } x = 0 \text{ and } y = 0, \end{cases} \tag{1}$$

where in the last case, we return 0 even though the result is undefined. We found that this version is about 39% faster than the function provided in the C standard math library. Finally, *SolTrack* treats all angles in radians, which avoids a computational overhead from constant conversion to and from degrees. The input and output can be easily converted from and to degrees at the beginning and end of the call respectively. A switch is available to the user to tell the routine that degrees are used for input and output if wanted.

2.1.1. Date, time and location

The desired date and time are provided as an input for the code in the common Gregorian calendar system. As is customary in astronomical calculations, these are converted to a Julian day (JD; *e.g.* Urban and Seidelmann, 2012), which provides a continuous time variable measured in days. The JD

[2]In fact, the library also provides routines for higher and lower accuracy than `sunpos_la()`.

is further converted to time in Julian days ($t_{\rm Jd}$) and centuries ($t_{\rm Jc}$) since the year 2000.0, which are required for input in many of the equations. The geographic location of the observer is provided as the longitude $l_{\rm obs}$ (where east from the Greenwich meridian is positive) and latitude $b_{\rm obs}$ (where the northern hemisphere is positive), in either radians or degrees.

2.1.2. Ecliptical coordinates

The ecliptic plane forms the base plane of the solar system, hence position calculations regarding orbital motions of the planets are usually performed in ecliptical coordinates. The geocentric position of the Sun is easily computed from the heliocentric position of the Earth in its orbit around the Sun in ecliptical coordinates by negating the ecliptical latitude and adding π (180°) to the ecliptical longitude.

The ecliptical longitude is computed as follows. The mean longitude (Simon et al., 1994) and mean anomaly (Chapront-Touze and Chapront, 1988) of the Sun are given by (when dropping unnecessary higher-order terms):

$$\begin{aligned} \lambda_0 &= 4.895063168 + 628.331966786 \cdot t_{\rm Jc} + 5.291838 \times 10^{-6} \cdot t_{\rm Jc}^2; \quad (2)\\ M &= 6.240060141 + 628.301955152 \cdot t_{\rm Jc} - 2.682571 \times 10^{-6} \cdot t_{\rm Jc}^2. \quad (3)\end{aligned}$$

The Sun's equation of the centre is the difference between the true and mean anomalies and between the true and mean longitudes of the Sun. It can be approximated using Meeus (1998, up to the sin 2M term):

$$\begin{aligned} C &= \left(3.34161088 \times 10^{-2} - 8.40725 \times 10^{-5} \cdot t_{\rm Jc} - 2.443 \times 10^{-7} \cdot t_{\rm Jc}^2\right) \sin M \\ &+ \left(3.489437 \times 10^{-4} - 1.76278 \times 10^{-6} \cdot t_{\rm Jc}\right) \sin 2M. \quad (4)\end{aligned}$$

The true longitude is then simply:

$$\lambda = \lambda_0 + C. \quad (5)$$

In order to apply a correction to the Sun's position for nutation and aberration, we need the longitude of the *Moon*'s mean ascending node Ω (Chapront-Touze and Chapront, 1988, up to second order):

$$\Omega = 2.1824390725 - 33.7570464271 \cdot t_{\rm Jc} + 3.622256 \times 10^{-5} \cdot t_{\rm Jc}^2. \quad (6)$$

The *nutation in longitude* is a periodic wobble of the Earth's rotation axis due to the presence of the Moon and can be described with sufficient accuracy by (Seidelmann, 1982, leading term only):

$$\Delta\psi = -8.338601 \times 10^{-5} \sin \Omega. \quad (7)$$

If an accurate distance is important, we compute the geocentric distance of the Sun in AU from the eccentricity of the Earth's orbit e (Simon et al., 1994, to second order) and the true anomaly ν:

$$\begin{aligned} e &= 0.016708634 - 4.2037 \times 10^{-5} \cdot t_{\rm Jc} - 1.267 \times 10^{-7} \cdot t_{\rm Jc}^2; \quad (8)\\ \nu &= M + C; \quad (9)\\ R &= 1.0000010178 \, \frac{1 - e^2}{1 + e \cos \nu}. \quad (10)\end{aligned}$$

In other cases, we simply assume a circular orbit:

$$R = 1.0000010178. \quad (11)$$

Annual aberration is the effect that the light from a celestial body seems to come slightly more from direction of the Earth's motion around the Sun, similar to the apparent motion of rain drops coming from the direction a car is moving into (but much less so, since the orbital speed of the Earth is much smaller than the speed of light). We use (Kovalevsky and Seidelmann, 2004):

$$\Delta\lambda = \frac{-9.93087 \times 10^{-5}}{R}. \quad (12)$$

We can now correct the longitude of the Sun for nutation and aberration, and arrive at the ecliptical longitude at the given instant:

$$\lambda = \lambda + \Delta\lambda + \Delta\psi. \quad (13)$$

Since we do not need extremely high accuracy, and since the Sun is (by definition) always near the ecliptic, for the ecliptical latitude of the Sun we simply use

$$\beta = 0. \tag{14}$$

Finally, we need the obliquity of the ecliptic ε, the angle between the Earth's rotation axis and the normal to the ecliptic plane, for coordinate transformations later on. However, we calculate it here because Ω is now available. We compute it from the mean obliquity ε_0 (Meeus, 1998) and the nutation in obliquity $\Delta\varepsilon$ (Seidelmann, 1982, leading term only):

$$\begin{aligned}
\varepsilon_0 &= 0.409092804222 - 2.26965525 \times 10^{-4} \cdot t_{\mathrm{Jc}} - 2.86 \times 10^{-9} \cdot t_{\mathrm{Jc}}^2; && (15)\\
\Delta\varepsilon &= 4.4615 \times 10^{-5} \cos\Omega; && (16)\\
\varepsilon &= \varepsilon_0 + \Delta\varepsilon. && (17)
\end{aligned}$$

2.1.3. Equatorial and parallactic coordinates

The equatorial coordinates *right ascension* α and *declination* δ are computed from the ecliptical coordinates (λ, β) using a standard coordinate transformation (*e.g.* Urban and Seidelmann, 2012):

$$\begin{aligned}
\tan\alpha &= \frac{\sin\lambda\,\cos\varepsilon - \tan\beta\,\sin\varepsilon}{\cos\lambda}; && (18)\\
\sin\delta &= \sin\beta\,\cos\varepsilon + \cos\beta\,\sin\varepsilon\,\sin\lambda, && (19)
\end{aligned}$$

where we use the `atan2()` function to compute the right ascension from Eq. 18.

The parallactic coordinate hour angle is the difference between the local sidereal time θ and the right ascension:

$$H = \theta - \alpha, \tag{20}$$

where

$$\begin{aligned}
\theta &= \theta_0 + \Delta\theta + l_{\mathrm{obs}}; && (21)\\
\theta_0 &= 4.89496121 + 6.300388098985 \cdot t_{\mathrm{Jd}} + 6.77 \times 10^{-6} \cdot t_{\mathrm{Jc}}^2; && (22)\\
\Delta\theta &= \Delta\psi \cdot \cos\varepsilon. && (23)
\end{aligned}$$

Here, θ is the local sidereal time, l_{obs} is the observer's geographical longitude (east of Greenwich is positive), θ_0 is the Greenwich mean sidereal time and $\Delta\theta$ corrects for nutation in right ascension, converting to *apparent* Greenwich sidereal time.

Together, hour angle and declination provide the coordinates needed for a *parallactic* mount.

2.1.4. Horizontal coordinates

We use the parallactic coordinates hour angle H and declination δ to compute the horizontal coordinates *azimuth* A and *altitude* h. This too is a well known coordinate transformation (*e.g.* Urban and Seidelmann, 2012):

$$\begin{aligned}
\sin h &= \sin\delta\,\sin b_{\mathrm{obs}} + \cos H\,\cos\delta\,\cos b_{\mathrm{obs}}; && (24)\\
\tan A &= \frac{\sin H}{\cos H\,\sin b_{\mathrm{obs}} - \tan\delta\,\cos b_{\mathrm{obs}}}, && (25)
\end{aligned}$$

where b_{obs} is the geographic latitude of the observer (northern hemisphere is positive). We use the `atan2()` function again to compute the azimuth from Eq. 25.

A variable that is often used instead of the altitude is the *zenith angle*, which is simply given by

$$\zeta = \frac{\pi}{2} - h. \tag{26}$$

2.1.5. Correction for atmospheric refraction

On the last part of their journey to the Earth's surface, the Sun's rays travel through the atmosphere of the Earth. The effect of the increasing density of the atmosphere causes refraction of the light rays towards the Earth's surface, so that they generally hit the surface somewhat earlier (nearer the Sun) than if the atmosphere had been absent. The result for an observer on the ground is that the altitude of the Sun h appears to be somewhat higher than we have just computed. The effect vanishes if the Sun is in the zenith, and has a maximum close to the horizon of about 0.5°, roughly the Sun's apparent diameter. For accurate positions, we need to take this effect into account, and because atmospheric refraction affects the altitude only, this is most easily done in *horizontal* coordinates.

We use a simple prescription by Saemundsson (1986), which, when expressed in radians, looks as follows:

$$\Delta h = 0.0002967 \cot \left(h + \frac{0.0031376}{h + 0.0892} \right) \left(\frac{T}{283} \right)^{-1} \left(\frac{P}{101} \right). \tag{27}$$

Here, cot() is the cotangent, h the uncorrected altitude, T the temperature in Kelvin and P the atmospheric pressure in kPa. For standard atmospheric values of $P = 101\,\mathrm{kPa}$ and $T = 283\,\mathrm{K}$ (10°C), the last two fractions can be ignored. We then *add* the quantity Δh to the uncorrected altitude h to account for the atmospheric refraction. Alternatively, we *subtract* Δh from the uncorrected zenith angle ζ.

SolTrack also offers refraction-corrected *parallactic* coordinates. These are obtained by transforming the corrected horizontal coordinates back to the parallactic system using the inverse transformation of Eqs. 24–25.

3. Performance of the *SolTrack* code

3.1. *Accuracy of* SolTrack

In order to assess the accuracy of our code, we need a comparison. We used the VSOP 87 code to compute very precise positions of the Sun, with an accuracy of 1.4×10^{-6} between the years 1900 and 2100 (Bretagnon and Francou, 1988). In order to correct for atmospheric refraction, we used an accurate model (Hohenkerk and Sinclair, 1985) that integrates the path of a light beam as it travels through the atmosphere for a given *apparent* sky position. This model is used in a slow, iterative procedure to solve the inverse problem, in order to find the apparent position for a given *true* sky position, as implemented in *libTheSky* (van der Sluys, 2002–2022). We assumed standard atmospheric pressure and temperature. The accuracy of the result will be limited by the detailed model for atmospheric refraction. In this section, we assume that the results from the combination of these two accurate models are exact, so that any deviation between it and *SolTrack* indicates an inaccuracy in our code. This assumption is validated by the fact that the detailed models are about three orders of magnitude more accurate than *SolTrack*.

The comparison between VSOP 87 and *SolTrack* was carried out for 100,000 random moments in the next 100 years (between 2017 and 2116) when the Sun is above the horizon in the Netherlands. We find that the error in position (either in parallactic or horizontal coordinates) is $0.0036 \pm 0.0042°$, or about 0.68% of the apparent diameter of the Sun. This is sufficient for solar tracking of HCPV systems under all conditions.

Figure 1 shows the positional error for 10,000 of the computed sky positions as a function of time over the next 100 years. We see that the accuracy hardly deteriorates over the coming century, so that the *SolTrack* algorithm can be used safely for at least the next 100 years. The figure also shows that most positions yield an accuracy that is better than 0.01°, while only a small fraction of points are less accurate than that. The latter group represents very low Sun positions, $\lesssim 2.5°$, where the correction for atmospheric refraction becomes less accurate. If these low-altitude points are ignored, on the grounds that very little energy will be generated at such Sun altitudes, the accuracy, and in particular its standard deviation, drop to $0.0030 \pm 0.0016°$.

The accuracy of the computed distance is $0.0017\% \pm 0.0029\%$, which results in an typical error in the computed solar power of 0.0058%. This will be completely negligible in a model where the uncertainty in weather conditions will have a much larger impact. The accuracy of the rise, transit and set times has been determined to be better than one second, while the corresponding azimuths and altitudes have an accuracy that is better than 10^{-3}°. The first uncertainty is far smaller than needed, since weather conditions provide a much larger source of uncertainty.[3] The second uncertainty is better than that for the position in general (since it is a one-dimensional problem, rather than a two-dimensional one).

3.1.1. *Comparison of* SolTrack*'s accuracy to other algorithms*

We have computed sky positions for the Sun for the same 100,000 random moments in next 100 years using PSA's *SunPos* (Plataforma Solar de Almería, 2013) and NREL's SPA (Reda and Andreas, 2004), and compared them to the VSOP87 positions. The results are displayed in Figures 2 and 3 and in Table 1.

When compared to *SunPos* routine, which is also lightweight and freely available, *SolTrack* is about 20 times more accurate in both horizontal and parallactic coordinates. Compared to the performance of

[3]Typically, the weather imposes an uncertainty of at least one minute on rise and set times.

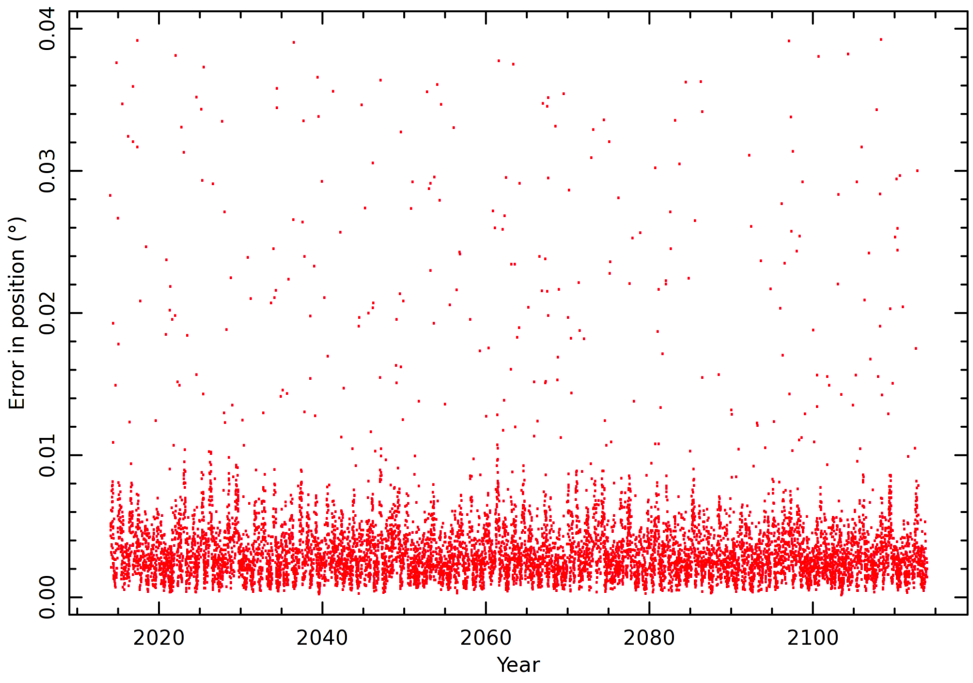

Figure 1: The accuracy of the *SolTrack* algorithm as a function of time for 10,000 random instances where the Sun is above the horizon in Arnhem, the Netherlands between 2017 and 2116. While for most cases, the accuracy is better than 0.01°, a relatively small number of data points indicate a larger deviation. All these are near sunrise or sunset (see Fig.2).

Table 1: Comparison of the accuracy, resulting loss of power w.r.t. optimal tracking with and without secondary optics (SO) between the *SunPos*, SPA and *SolTrack* routines.

		SunPos	**SPA**	***SolTrack***
Accuracy	mean	0.073°	0.0023°	0.0036°
in position	st.dev.	0.091°	0.0036°	0.0042°
	relative	20.2	0.64	1.00
Accuracy	mean	—	0.0017%	0.0029%
in distance	st.dev.	—	0.0011%	0.0021%
in power	mean	—	0.0034%	0.0058%

the code to the NREL routine SPA, which is more elaborate and has a more restricted licence, SPA is 36% more accurate than *SolTrack* when comparing horizontal coordinates. However, SPA does not offer refraction-corrected parallactic coordinates, so that our code is about 20 times more accurate when a parallactic mount is used. For this comparison we computed only the Sun's position in SPA, no rise and set times or incident radiation. An overview of the comparison between the three codes can be found in Table 1.

3.2. Speed of SolTrack

In order to benchmark the *SolTrack* C code, we timed runs where 10^7 (ten million) positions were computed on a single 2.67 GHz CPU core of a normal laptop computer. We used the `GCC` compiler (v6.2 The GCC project, 2022) with `-O2` optimisation and the `taskset` Linux utility to force the program to always run on the same core (The Linux Foundation, 2022). In these speed-benchmark runs we generated the date and time randomly, in order to avoid an I/O overhead from reading these data from disc. We did not save the computed results for the same reason.

We performed ten such runs and found a mean run time and standard deviation (averaged over the ten runs of 10^7 calls to the *SolTrack* routine each) of 7.331 ± 0.024 s. We then repeated the test, but without actually calling the *SolTrack* routine, which took 0.860 ± 0.012 s (again the mean of ten runs of 10^7 calls each). This number represents the overhead from the code that *calls SolTrack* rather than from *SolTrack* itself, *e.g.* from starting the code and generating the random date and time. Hence, we determine the CPU time for 10^7 calculations of the Sun's position on the used CPU core as the difference

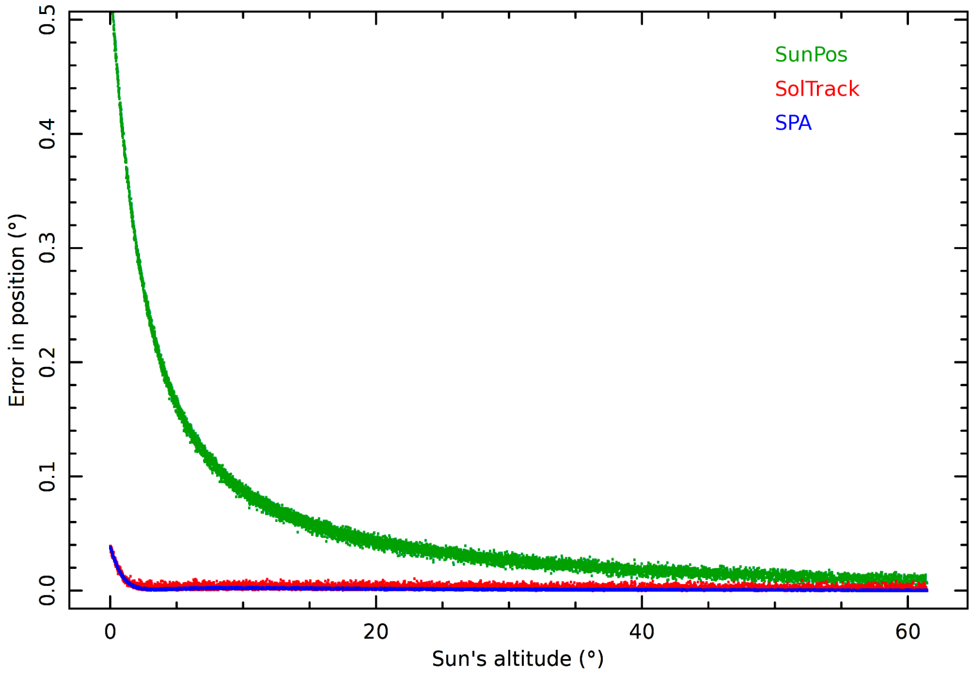

Figure 2: A comparison of the accuracies of PSA's *SunPos* (green/grey, at the top), *SolTrack* (red/grey, near the bottom) and SPA (blue/black, at the bottom), as a function of altitude for 10,000 random instances where the Sun is above the horizon in Arnhem, the Netherlands. The data points for *SolTrack* partially overlap with those for SPA (see Fig. 3 for a clearer view). The largest difference is found between *SunPos* and the other two codes, especially for low altitudes, mainly due to the lack of correction for atmospheric refraction in *SunPos*.

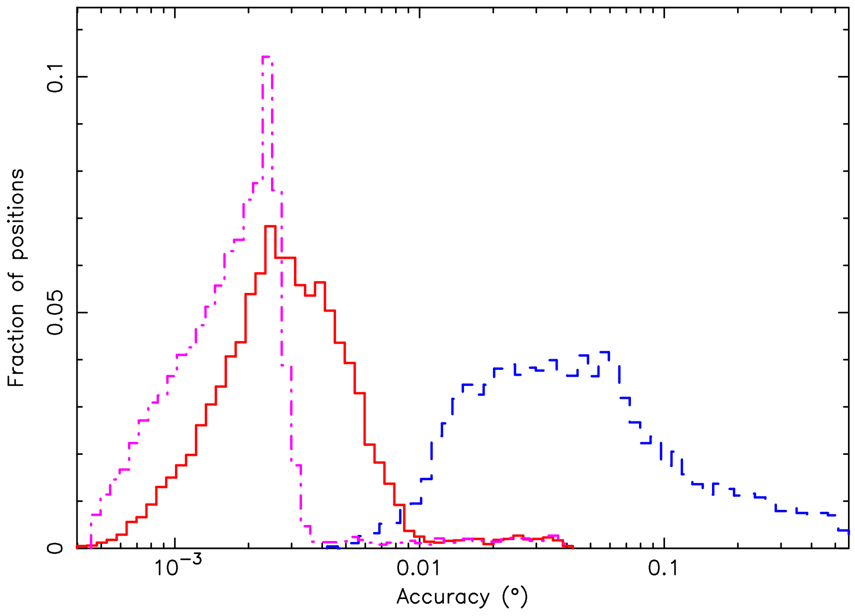

Figure 3: Three histograms comparing the accuracies of SPA (dash-dotted/purple line, to the left), *SolTrack* (solid/red line, in the middle) and PSA's *SunPos* (dashed/blue line, to the right), for 10,000 random instances where the Sun is above the horizon in Arnhem, the Netherlands. Note the logarithmic scale on the horizontal axis.

Table 2: Comparison of the CPU times for the calculation of 10^7 positions between modes where neither the distance or refraction correction for equatorial coordinates are computed, or where one or both are computed.

Distance?		×	✓	×	✓
Ref corr eq.?		×	×	✓	✓
CPU time	mean	6.471 s	6.987 s	8.160 s	8.686 s
	st.dev.	0.027 s	0.028 s	0.022 s	0.030 s
	relative	1.000	1.080	1.261	1.342

Table 3: Comparison of the CPU times for the calculation of 10^7 positions between *SunPos*, SPA and *SolTrack* routines. The last column shows the results for *SolTrack* with the additional refraction correction in equatorial coordinates.

		SunPos	**SPA**	***SolTrack***	***SolTrack* + eq.**
CPU	mean	6.632 s	160.78 s	6.471 s	8.160 s
time	st.dev.	0.026 s	0.45 s	0.027 s	0.022 s
	relative	1.025	19.70	1.000	1.261

of these two numbers, *i.e.* 6.471 ± 0.027 s, or about 1.5×10^6 calls per second.[4] These numbers are valid for the default mode of *SolTrack* where the refraction correction is computed only for horizontal coordinates and the Sun-Earth distance is not computed. In these tests we only computed positions of the Sun; we did not determine rise, transit or set times.

In addition, we computed the same numbers, *i.e.* the mean CPU time for ten runs of 10^7 *SolTrack* calls each, corrected for the overhead determined from runs without calls to *SolTrack* for the other three modes of *SolTrack* where the distance is computed as well and/or the equatorial coordinates are corrected for atmospheric refraction as well. The results are shown in Table 2. We find that computing the distance adds about 8% to the CPU time, while correcting equatorial coordinates for refraction costs 26% extra.

3.2.1. *Comparison of* SolTrack*'s computational speed to other algorithms*

We compare the CPU times of *SolTrack* to those of *SunPos* (Plataforma Solar de Almería, 2013) and SPA (Reda and Andreas, 2004) in Table 3. The table shows that *SolTrack* is about 2.5% faster than *SunPos* if the refraction correction is only applied to horizontal coordinates, and 23% slower if this correction is also applied for equatorial coordinates. Hence, *SolTrack*'s performance in computational speed is similar to that of *SunPos*, despite the much greater accuracy of our code (see Sect. 3.1).

When comparing *SolTrack* to NREL's PSA code, our code is almost a factor of 20 faster if the refraction correction is limited to horizontal coordinates, and nearly 16 times faster if *SolTrack* corrects equatorial coordinates for refraction as well, an option that does not exist in PSA. As we saw in the previous section, PSA gives more accurate results for its higher computational costs, albeit not by more than an order of magnitude.

4. Summary and conclusions

In this paper, we presented *SolTrack*, a free, fast and accurate code to compute the position of the Sun and do related calculations. Due to its open-source licence and the fact that the code is written in plain C and Python, it can be used by a wide range of users on a wide range of computing platforms and for a wide range of purposes, be it scientific, commercial or otherwise. The mean accuracy over the period 2017–2116 is 0.0036 ± 0.0042°, and if instances where the Sun is less than 2.5° above the horizon are omitted, this improves to 0.0030 ± 0.0016°. *SolTrack* can also compute rise, transit and set times to an accuracy better than 1 second. *SolTrack* can compute millions of positions per second on a typical modern CPU, and as such is capable of accurately tracking a solar-concentration system on a low-spec platform, as well as perform many calculations in a yield model (*e.g.* using a Monte Carlo method).

When compared to the two codes that are currently used in the field of (high-)concentration photovoltaics ((H)CPV) and concentrated solar power (CSP), it is comparable in its open licence and computing speed to the first (PSA) while being about 20 times more accurate, and only slightly less accurate than the other (SPA) whilst about 20 times faster and much freer in its licence.

[4]Scaled with the clock speed of the core, a 1.7 kHz CPU would be needed in order to compute the Sun's position every second if this were its only task.

References

Bretagnon, P., Francou, G., 1988. Planetary theories in rectangular and spherical variables - VSOP 87 solutions. Astronomy & Astrophysics 202, 309–315.

Chapront-Touze, M., Chapront, J., 1988. ELP 2000-85 - A semi-analytical lunar ephemeris adequate for historical times. Astronomy & Astrophysics 190, 342–352.

Free Software Foundation, 2007. The GNU Lesser General Public License. URL: https://www.gnu.org/copyleft/lesser.html.

Hohenkerk, C., Sinclair, A., 1985. The computation of angular atmospheric refraction at large zenith angles. NAO Technical Note 63.

Kovalevsky, J., Seidelmann, P.K., 2004. Fundamentals of Astrometry.

Meeus, J., 1998. Astronomical algorithms.

Plataforma Solar de Almería, 2013. SunPos URL: http://www.psa.es/sdg/sunpos.htm.

Reda, I., Andreas, A., 2004. Solar position algorithm for solar radiation applications. Solar Energy 76, 577–589. doi:10.1016/j.solener.2003.12.003.

Saemundsson, T., 1986. Atmospheric Refraction. Sky & Telescope 72, 70.

Seidelmann, P.K., 1982. 1980 IAU theory of nutation - The final report of the IAU Working Group on Nutation. Celestial Mechanics 27, 79–106. doi:10.1007/BF01228952.

Simon, J.L., Bretagnon, P., Chapront, J., Chapront-Touze, M., Francou, G., Laskar, J., 1994. Numerical expressions for precession formulae and mean elements for the Moon and the planets. Astronomy & Astrophysics 282, 663–683.

The GCC project, 2022. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/.

The GNU Project, 2022. GNU binutils. URL: https://sourceware.org/binutils/.

The Linux Foundation, 2022. util-linux. URL: https://www.kernel.org/pub/linux/utils/util-linux/.

The Valgrind Developers, 2022. Valgrind. URL: http://www.valgrind.org.

Urban, S.E., Seidelmann, P.K., 2012. Explanatory Supplement to the Astronomical Almanac (3rd Edition). University Science Books.

van der Sluys, M., 2002–2022. libTheSky. URL: http://libthesky.sf.net.

van der Sluys, M., van Kan, P., 2014–2022. SolTrack URL: http://soltrack.sf.net.

van der Sluys, M., van Kan, P., Sonneveld, P., 2015. CPV in the built environment, in: American Institute of Physics Conference Series, p. 080003. doi:10.1063/1.4931544.

Wikipedia, 2019. Atan2 — wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Atan2. [Online; accessed 2019-02-18].