Comparison of Earth-Centered, Earth-Fixed to Geodetic Coordinate Conversion Algorithms

Steven D. Ward
planet36@gmail.com

Version:
2020-02-05

Keywords:
coordinate conversion, ECEF, geodetic, Cartesian, geocentric, geodesy, latitude, algorithms, WGS 84 ellipsoid

ABSTRACT: *The conversion of Earth-Centered, Earth-Fixed (ECEF) coordinates to geodetic latitude is inexact, and many algorithms for it have been developed in recent decades.*
This paper compares the performance, as measured by accuracy and speed, of approximately 40 such algorithms using input locations ranging from the centroid of the World Geodetic System 1984 (WGS 84) ellipsoid to beyond lunar orbit. Other factors such as code size, copyright, and licensing are presented, and suggestions of specific algorithms are made for various simulation use cases.

1. Introduction

The IEEE Standard for Distributed Interactive Simulation (DIS) [1] uses a right-handed, geocentric Cartesian (also known as ECEF) coordinate system, whose origin is the WGS 84 reference frame [2], to identify locations in a simulated world.

To convert from a human-friendly geodetic coordinate to ECEF coordinate, one would use exact calculations. However, the inverse conversion is inexact – it can only be done by iteration or approximation.

Many ECEF-to-geodetic algorithms have been developed in recent decades, yet the DIS standard gives no suggestion of which one to use. This paper will compare the performance, as measured by accuracy and speed, of many such algorithms. Additionally, the software copyright, license, and size are considered in the suggestions.

1.1 Notation

φ, λ, h = geodetic coordinate: latitude, longitude, height (above ellipsoid)

X, Y, Z = ECEF coordinate; $W = \sqrt{X^2 + Y^2}$

The WGS 84 reference frame is the ellipsoid used. Height below the ellipsoid is negative.

2. Algorithms

The algorithms evaluated in this paper are publicly available either in peer reviewed journals/publications, software libraries, or commonly-known naïve formulas. In the interest of brevity, the sources of all the algorithms may be found in [3], not in this paper. Formulas and derivations of naïve algorithms may also be found in [3].

All algorithms were programmed into C++, either from interpreting mathematics in the published papers, copying from existing software libraries, or translating from another programming language.

Code written by this author is released under the Open Software License 3.0 (OSL-3.0) [4].

Algorithm Name	Algorithm Author	Code Copyright	Original Code Language	Code License
borkowski_1989	K.M. Borkowski	Steven Ward	None	OSL-3.0
bowring_1976	B.R. Bowring	Steven Ward	None	OSL-3.0
bowring_1985	B.R. Bowring	Steven Ward	None	OSL-3.0
bowring_toms_1995	B.R. Bowring, R. Toms	Steven Ward	None	OSL-3.0
fukushima_1999	Toshio Fukushima	Steven Ward	None	OSL-3.0
fukushima_2006	Toshio Fukushima	Toshio Fukushima	Fortran	Unknown
GeographicLib	Charles F. F. Karney	Charles F. F. Karney	C++	MIT/X11
geotransformCpp	R. Toms	SEDRIS	C++	"This software was developed for use by the United States Government with unlimited rights."
gersten_1961	R. H. Gersten	Steven Ward	None	OSL-3.0
halley	Steven Ward	Steven Ward	None	OSL-3.0
heikkinen_1982	M. Heikkinen	Steven Ward	None	OSL-3.0
householder	Steven Ward	Steven Ward	None	OSL-3.0

Algorithm Name	Algorithm Author	Code Copyright	Original Code Language	Code License
jat_spacetime_geodetic	O. Montenbruck, E. Gill	NASA	Java	NASA Open Source Agreement
jones_2002	C. Jones	Steven Ward	None	OSL-3.0
ligas_2011_I	M. Ligas, P. Banasik	Steven Ward	None	OSL-3.0
lin_wang_1995	Kuo-Chi Lin, Jie Wang	Steven Ward	None	OSL-3.0
long_1974	S. Long	Steven Ward	None	OSL-3.0
naive_I	Steven Ward	Steven Ward	None	OSL-3.0
naive_II	R. Hirvonen, H. Moritz	Steven Ward	None	OSL-3.0
newton_raphson	Steven Ward	Steven Ward	None	OSL-3.0
olson_1996	D.K. Olson	"U.S. Government work, U.S. copyright does not apply."	C	Public Domain
openglobe	Cozzi and Ohlarik	Cozzi and Ohlarik	C#	MIT
ozone_1985	M. Ozone	Steven Ward	None	OSL-3.0
paul_1973	M.K. Paul	Steven Ward	None	OSL-3.0
pollard_2002_ht	J. Pollard	Steven Ward	None	OSL-3.0
pollard_2002_newton	J. Pollard	Steven Ward	None	OSL-3.0
pollard_2002_simple	J. Pollard	Steven Ward	None	OSL-3.0
schroder	Steven Ward	Steven Ward	None	OSL-3.0
sedris	Cameron Kellough?	SEDRIS	C++	"This software was developed for use by the United States Government with unlimited rights."
shu_2010	C. Shu, et al.	Steven Ward	None	OSL-3.0
sofair_1993	I. Sofair	Steven Ward	None	OSL-3.0
sofair_2000	I. Sofair	Steven Ward	None	OSL-3.0
sudano_1997	J. Sudano	Steven Ward	None	OSL-3.0
turner_2013	J. Turner, T. Elgohary	Steven Ward	None	OSL-3.0
vermeille_2004	H. Vermeille	Steven Ward	None	OSL-3.0
vermeille_2011	H. Vermeille	Steven Ward	None	OSL-3.0
wu_2003	Y. Wu, et al.	Steven Ward	None	OSL-3.0
zhang_2005	CD. Zhang, et al.	CD. Zhang, et al.	MATLAB	Unknown
zhu_1993	Jijie Zhu	Steven Ward	None	OSL-3.0

Table 1 Algorithm intellectual property data

3. Testing Methodology

3.1 Test Environment

The algorithms were tested on a workstation with a 64-bit server-class CPU first sold in Q2 2016, Linux kernel 5.3.0, g++ 9.2.1, and glibc 2.30. The compiler optimization "-O3" was used [5]. Double precision floating point types and standard math functions were used.

3.2 Input Data

The algorithms must be given ECEF coordinates as input. But to control the distribution and shape of the input ECEF coordinates, five sets of geodetic coordinates were chosen first. The distribution of these sets is roughly uniformly distributed near h=0, ranging from -90° to 90° in latitude.

The five sets of input points, hereafter called regions, vary by increasing the range of ellipsoid heights and decreasing the density of the points. It is not necessary to vary the longitude because the distance from the Z-axis (denoted as W) is used in the conversions.

The input points were aggregated and deduplicated, resulting in approximately 8.6 million unique points.

None of the points reached inside the evolute of the WGS 84 ellipsoid, for reasons described in the next section.

Additionally, the points do not extend far beyond the lunar apogee because of limited applicability to most simulation scenarios.

The input geodetic coordinates were then converted to ECEF (which is an exact conversion), and then given as input to each algorithm, as shown in **3.4**.

Region	φ min	φ max	Δφ	*h* min	*h* max	Δ*h*	Approx. num. unique points
0	-90°	90°	0.0001°	0 m	0 m		1,800,000
1	-90°	90°	0.001°	-100 m	1000 m	100 m	2,200,000
2	-90°	90°	0.01°	-10,000 m	100,000 m	1000 m	2,000,000
3	-90°	90°	0.1°	-1,000,000 m	10,000,000 m	10,000 m	2,000,000
4	-90°	90°	1°	-5,000,000 m	500,000,000 m	100,000 m	900,000

Table 2 Input data regions

The following figures show plots of the input points. Orange dots are the points to be converted, dashed magenta lines are the W- and Z-axes, and the solid blue line is the WGS 84 ellipsoid.

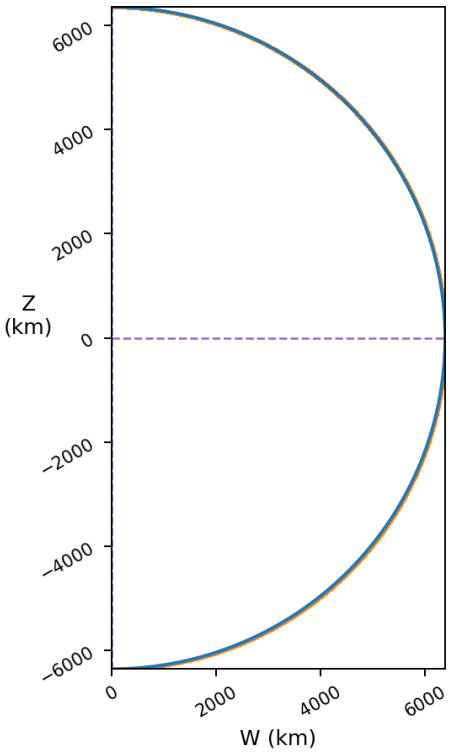

Figure 1 Input region 1

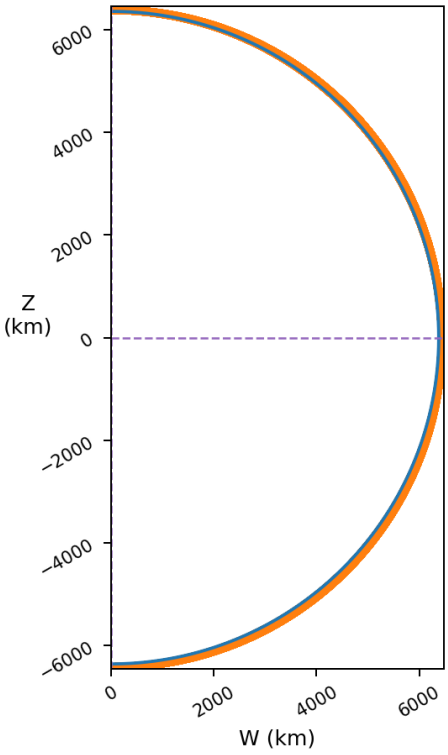

Figure 2 Input region 2

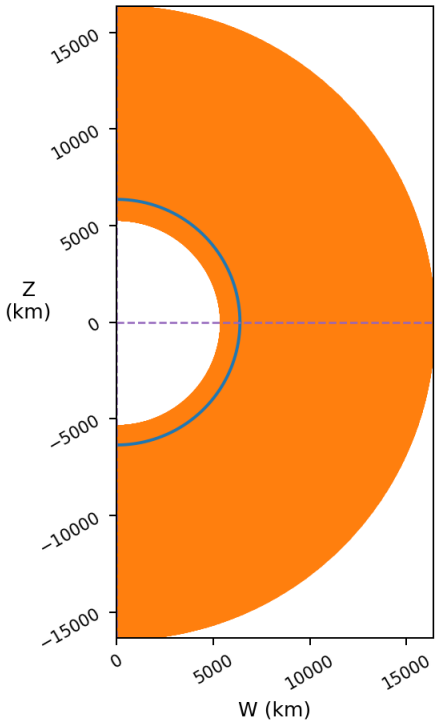

Figure 3 Input region 3

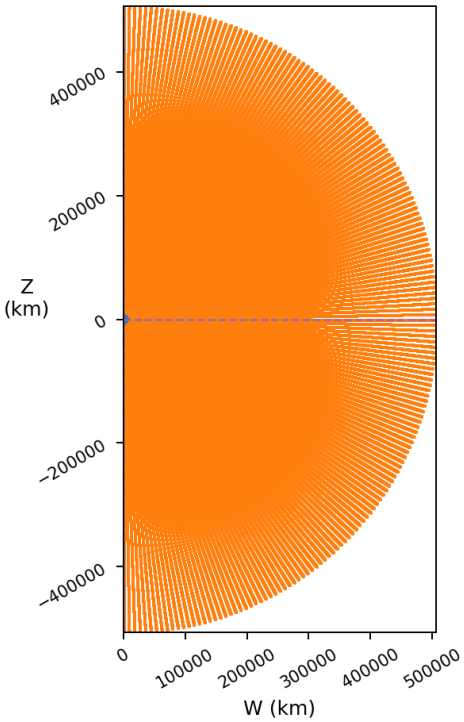

Figure 4 Input region 4

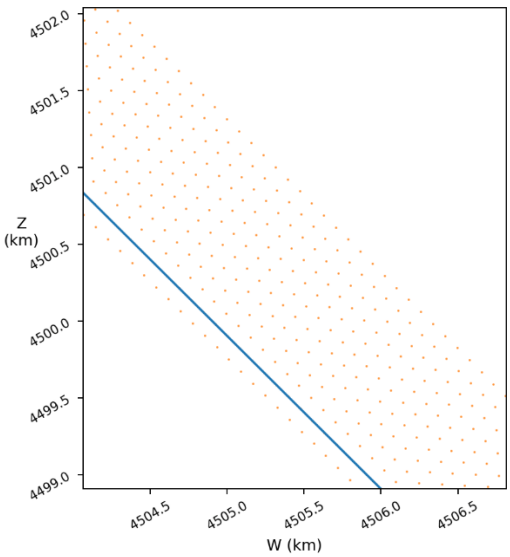

Figure 5 Input region 1 (zoomed)

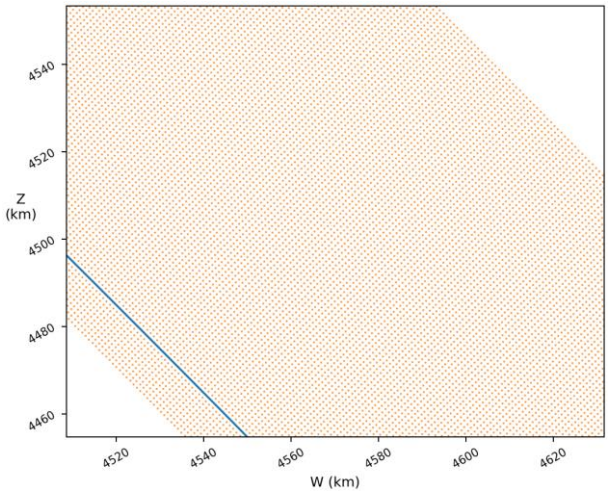

Figure 6 Input region 2 (zoomed)

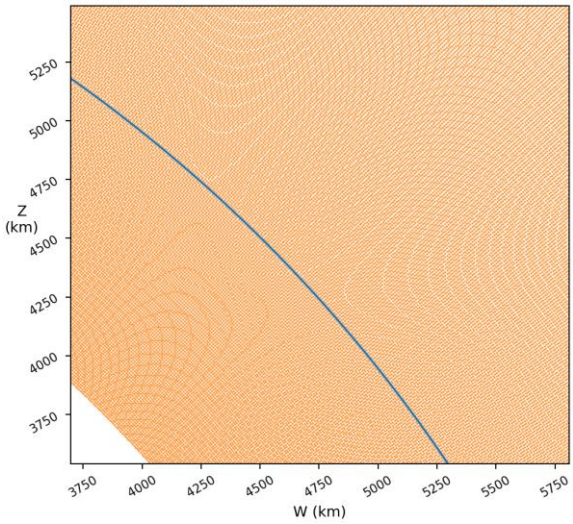

Figure 7 Input region 3 (zoomed)

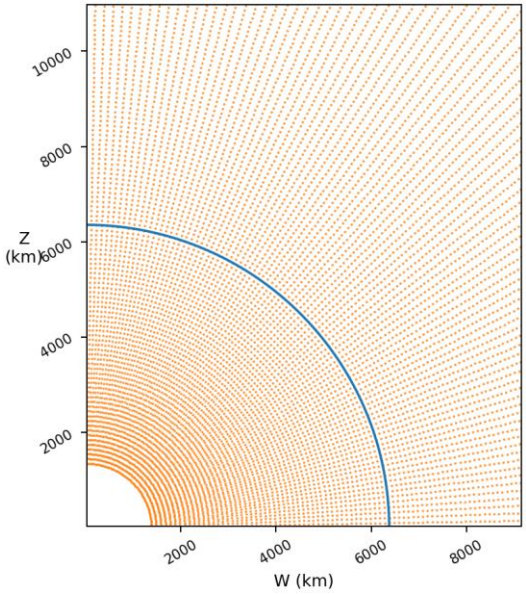

Figure 8 Input region 4 (zoomed)

3.2.1 Toward the Center of the Earth

A separate accuracy test using input points approaching the centroid of the WGS 84 ellipsoid at φ=45° was done. Most algorithms became unacceptably inaccurate within 100km from the centroid. Every algorithm failed with points on or inside the evolute of the ellipsoid except two: GeographicLib [6] and vermeille_2011 [7].

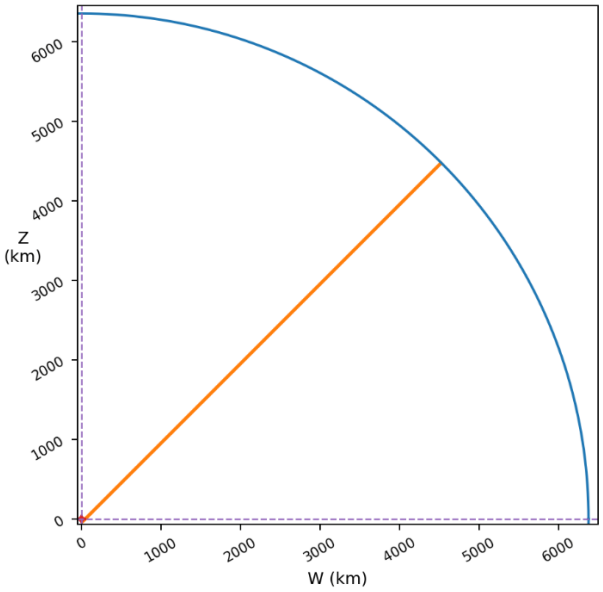

Figure 9 Input points within the ellipsoid, φ=45°deg

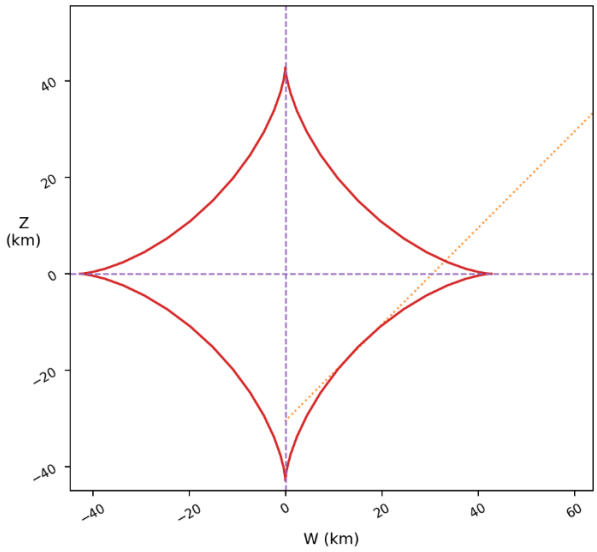

Figure 10 Input points near evolute (red) of the ellipsoid, φ=45°deg

3.3 Iteration Limit

For iterative algorithms, the number of iterations was limited to 2, regardless of the latitude difference computed in each iteration. As shown later in the accuracy results, many fast converging algorithms reach nanometer accuracy within two iterations. If the iteration limit was higher, a threshold for the latitude difference would have to be chosen.

3.4 Accuracy Measurement

The result of the ECEF-to-geodetic conversion was converted to an ECEF coordinate (which is an exact conversion), and the Euclidian distance between the input ECEF coordinate and result ECEF coordinate was taken to be the error of the algorithm.

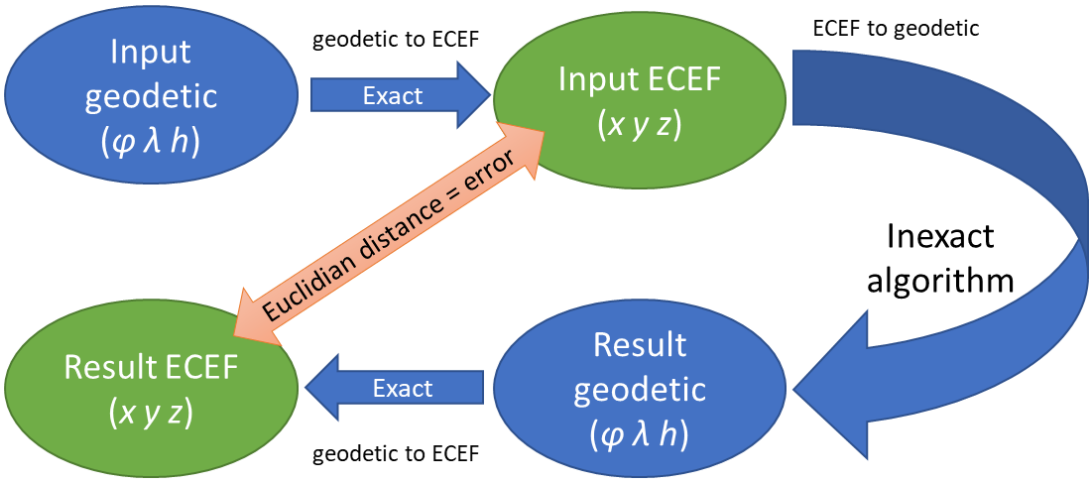

Figure 11 Flowchart of conversion of input data

3.5 Speed Measurement

The total elapsed time of each algorithm to convert all the input data (excluding the accuracy calculations) was measured using a standard C++ steady clock [8].

The performance tests were run 2001 times, and the order of the algorithms was shuffled each time. The median elapsed time was used as the measurement of each algorithm's speed.

3.6 Lines of Code

The number of source code lines includes the entire algorithm implementation and all comments. Algorithms originating from existing software libraries were not significantly modified, and all functions have the same signature, common declarations, and longitude calculations.

4. Performance Results

Algorithms with "*n=1*" or "*n=2*" in their name are iterative, where *n* is the number of iterations. All other algorithms are closed form approximations.

Algorithms with "*custom h*" in their names have a custom height formula, distinct from the "common" height formula. Both height formulas were tested, and in all cases, the custom height formula was faster but less accurate than the common height formula.

Some algorithms could not be translated into working code. Mean distance errors greater than 10 meters were ignored. The standard deviation of the median time per function call did not exceed 3.0 nanoseconds.

The result data values are color-coded by Microsoft Excel's conditional formatting colors scales. For all data values, lower is better.

Algorithm name	Mean distance error (m)	Median time/call (ns)	Lines of code
borkowski_1989	1.19E-07	120	51
bowring_1976 *(n=1)*	2.54E-02	52.5	23
bowring_1976 *(n=2)*	2.07E-09	67	31
bowring_1985 *(n=1)*	5.34E-07	63	27
bowring_1985 *(n=2)*	2.00E-09	78	35
bowring_toms_1995 *(n=1)*	2.51E-03	56.3	47
bowring_toms_1995 *(n=2)*	2.05E-09	70.6	55
fukushima_1999 *(n=1)*	Couldn't get it to work		118
fukushima_1999 *(custom h) (n=1)*	Couldn't get it to work		118
fukushima_2006 *(n=1)*	3.94E-05	53	42
fukushima_2006 *(n=2)*	Overflow at h>8,000km		42
geographiclib	1.04E-09	147.5	106
geographiclib *(custom h)*	2.97E-09	123.5	106
geotransformCpp	1.18E-04	107.3	208
geotransformCpp *(custom h)*	1.21E-04	84.7	208
gersten_1961	Couldn't get it to work		40
halley *(n=1)*	8.25E-02	92.3	42
halley *(n=2)*	9.30E-10	135.4	42
halley_quick *(n=1)*	8.25E-02	87.6	48

Algorithm name	Mean distance error (m)	Median time/call (ns)	Lines of code
halley_quick *(n=2)*	2.80E-09	120.4	48
heikkinen_1982	1.92E-09	123.2	56
heikkinen_1982 *(custom h)*	2.57E-09	119.6	56
householder *(n=1)*	8.25E-02	93.2	42
householder *(n=2)*	9.33E-10	137	42
householder_quick *(n=1)*	8.25E-02	88.6	48
householder_quick *(n=2)*	2.79E-09	122	48
jat_spacetime_geodetic	8.45E-03	138.7	44
jones_2002 *(n=1)*	1.18E+01	53.3	60
ligas_2011_I *(n=1)*	3.84E-02	61.2	87
ligas_2011_I *(n=2)*	1.26E-08	75.3	87
lin_wang_1995 *(n=1)*	4.30E-07	83	56
lin_wang_1995 *(n=2)*	2.29E-09	102.1	56
lin_wang_1995 *(custom h) (n=1)*	1.13E-04	79	56
lin_wang_1995 *(custom h) (n=2)*	2.46E-09	97.9	56
long_1974	Couldn't get it to work		27
naive_I *(n=1)*	5.27E+00	63.4	32
naive_I *(n=2)*	3.61E-03	89.8	32
naive_II *(n=1)*	4.70E+00	63.9	26
naive_II *(n=2)*	3.04E-03	91.6	26
newton_raphson *(n=1)*	1.52E-01	85.1	40
newton_raphson *(n=2)*	9.47E-10	121.8	40
newton_raphson_quick *(n=1)*	1.52E-01	80.8	46
newton_raphson_quick *(n=2)*	2.93E-09	104.8	46
olson_1996	1.38E-09	70.8	68
olson_1996 *(custom h)*	2.67E-09	42.9	68
openglobe	1.39E-05	110.3	189
ozone_1985	2.33E-07	127.6	67
paul_1973	Couldn't get it to work		65
pollard_2002_ht *(n=1)*	Couldn't get it to work		53
pollard_2002_naive *(n=1)*	8.68E-02	62.2	32
pollard_2002_naive *(n=2)*	4.29E-04	76.5	40
pollard_2002_newton *(n=1)*	5.34E-07	69.4	41
pollard_2002_newton *(n=2)*	1.80E-09	90.4	55
schroder *(n=1)*	3.13E-01	92.4	42
schroder *(n=2)*	9.51E-10	135.6	42
schroder_quick *(n=1)*	3.13E-01	87.9	48
schroder_quick *(n=2)*	3.04E-09	120.3	48
sedris	7.86E-05	79.4	584
sedris *(custom h)*	1.06E-04	58.8	584
shu_2010 *(n=1)*	1.99E-07	63.5	45
shu_2010 *(n=2)*	2.35E-09	74	45
shu_2010 *(custom h) (n=1)*	5.22E-05	59.9	45
shu_2010 *(custom h) (n=2)*	3.03E-09	70.9	45
sofair_1993	Couldn't get it to work		49

Algorithm name	Mean distance error (m)	Median time/call (ns)	Lines of code
sofair_2000	6.55E+00	146.6	50
sudano_1997	Couldn't get it to work		62
turner_2013	Couldn't get it to work		85
vermeille_2004	1.35E-09	144.6	31
vermeille_2011	1.35E-09	156	84
vermeille_2011 *(custom h)*	3.41E-09	133.7	84
wu_2003 *(n=1)*	1.12E+02	79.1	91
wu_2003 *(n=2)*	4.59E-04	89.7	91
zhang_2005	2.41E-09	135.5	81
zhu_1993	4.84E-09	117.4	36

Table 3 Algorithm Performance Results

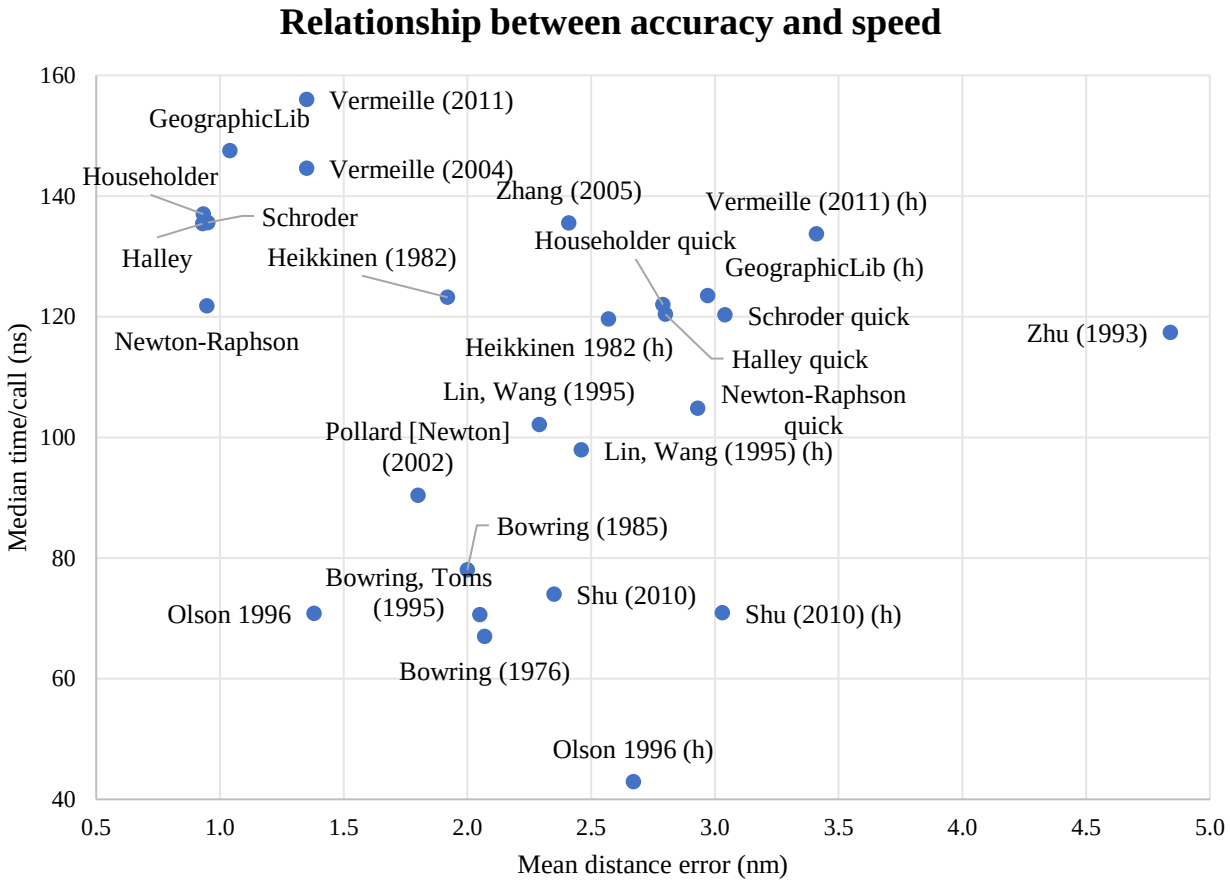

Figure 12 Scatter plot of speed vs accuracy for select algorithms

Figure 12 shows a scatter plot of speed versus accuracy for algorithms with accuracy better than 10 nanometers. Labels appended with "(h)" denote the variant of the algorithm that uses its custom height formula instead of the common height formula. All iterative algorithms use 2 iterations.

5. Recommendations

For the given input data, over a dozen algorithms yielded nanometer accuracy, which should be enough for everyone.

The **olson_1996** *(custom height)* [9] algorithm is recommended for ECEF-to-geodetic coordinate conversion. It was the fastest algorithm tested, of an acceptable size, and, being in the public domain, may be copied freely.

When using an iterative algorithm, doing 2 iterations is recommended.

If converting ECEF coordinates near the center of the Earth is required, the **GeographicLib** [6] software library by Karney is recommended. It lists features such as conversion between ECEF, geodetic, UTM/UPS, and MGRS coordinates, as well as geoid height, geomagnetic, and geodesic calculations.

6. References

[1] IEEE Standard for Distributed Interactive Simulation – Application Protocols; IEEE Std. 1278.1™-2012

[2] National Geospatial-Intelligence Agency (NGA) Standardization Document; Department of Defense World Geodetic System 1984; Its Definition and Relationships with Local Geodetic Systems; 2014-07-08; Version 1.0.0

[3] https://github.com/planet36/ecef-geodetic

[4] “The Open Software License 3.0 (OSL-3.0)”. https://opensource.org/licenses/OSL-3.0

[5] “Options That Control Optimization”. https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[6] GeographicLib by Charles F. F. Karney. https://geographiclib.sourceforge.io/

[7] Vermeille, H. J Geod (2011) 85: 105. https://doi.org/10.1007/s00190-010-0419-x

[8] “std::chrono::steady_clock”. https://en.cppreference.com/w/cpp/chrono/steady_clock

[9] Olson, D. K., “Converting Earth-centered, Earth-fixed coordinates to geodetic coordinates,” in IEEE Transactions on Aerospace and Electronic Systems, vol. 32, no. 1, pp. 473-476, Jan. 1996. doi: 10.1109/7.481290

Author Biography

STEVEN WARD is a software developer in Orlando, Florida.