Interoperability across Implementation: the libdrdc Data Standards Library

Dave Erickson*
Defence Research and Development Canada
PO Box 4000, Station Main, Medicine Hat,
Canada
david.erickson@drdc-rddc.gc.ca

Abstract*— This paper presents the libdrdc data standards library containing internal nomenclature, definitions, units of measure, coordinate reference frames, and representations for use in autonomous systems research. This library is a configurable, portable C / C++ / Object Oriented C library developed to be independent of software middleware, system architecture, processor, or operating system. It is designed to use the automatically-tuned linear algebra suite (ATLAS) and Basic Linear Algebra Suite (BLAS) and port to firmware and software. The library goal is to unify data collection and representation for various systems and communication protocols and to provide a common Application Programmer Interface (API) for research projects at all scales. The library supports multi-platform development and currently works on Windows, Unix, GNU/Linux, and Real-Time Executive for Multiprocessor Systems (RTEMS). *This library is available online under the Lesser GNU Public License (LGPL) version 2.1.

***Index Terms*— autonomy, applicaiton programmer interface, automatically tuned linear algebra linear algebra suite, libraries, representations, robotics, software, standards, teleoperation, units of measure, unmanned ground vehicle, ATLAS, API, BLAS, JAUS, ROV, RTEMS, UGV**

I. INTRODUCTION

This paper presents the libdrdc data standards library. This library contains the functionality and data representation conventions for global/local pose, position, orientation, manipulator link parameters, global time, homogeneous transforms using BLAS, unified time standard and epoch conversions, fixed-point mathematics, and exception handling compartmentalized into a single library and development project. The libdrdc library is a configurable C / C++ / Object Oriented C library providing a native set of data representations for units of measure and local and global coordinate reference frames. The application progammer interface (API) is C++-wrapped C functions designed for speed and size across implementation scales. The library uses the automatically-tuned linear algebra suite (ATLAS) [1] for performance. The library has been tested on Windows, Unix, GNU/Linux, and RTEMS operating systems using the PowerPC, Intel x86 series, and ARM processors in 16-, 32-, and 64-bit CPU architectures. The library goal is to free researchers to focus on data outcomes rather than on data collection handling.

This work was supported by DND through DRDC

D. Erickson is with the Autonomous Intelligent Systems Section, Defence Research and Development Canada, PO Box 4000, Station Main, Medicine Hat, Canada david.erickson@drdc-rddc.gc.ca

Current autonomous systems research requires the amalgamation of many complex systems. In general, large scale robotics projects involve some COTS, some open source, and some in-house software development to make a complete system of systems. For example, DRDC Suffield has many systems delivered and under development employing various operating systems or architectures including: ANCAEUS; VCS; MIRO; RTEMS; GNU/Linux; Player/Stage; Vortex; CMX; VxWorks; and Microsoft Windows amongst others. The use of any two of these software architectures together in the same project would require some thought in the conversion and cooperation of various modules and components. Data collected from these architectures would require detailed explanation for analysis by a third party. A single point of truth, a library employing uniform data representation conventions and data transformations, adopted across all robotic system elements is one data confusion mitigation strategy.

DRDC began developing libdrdc in 2005. The reasons for the selection of the standards reported in this implementation are many: some are general conventions, most were chosen in order to be compliant with Joint Architecture for Unmanned Systems (JAUS) [2] representations, and a few are arbitrary based on technical reasons. Whatever their origin, it is important to adopt comprehensive standards in order to reap the benefit of interoperability. Data exchange without common semantics leads to confusion and inefficiency, and is unacceptable in a research program that relies on a considerable amount of systems complexity and dispersive employment. All standard conventions outlined are open and explained in detail in the documentation. The standards themselves are open to discussion and improvement if that is determined by stakeholders.

This library is intended to be separate, transparent, open source, and multi-platform with extensive routines designed to support distributed, embedded unmanned ground vehicle (UGV) applications at any scale. The aim is to be agnostic in terms of operating systems, robot control architectures, middleware, communication protocol, processors, and so on. The functionality is limited to data representations and transformations and conversions. The library API abstracts quaternion orientations, Universal Transverse Mercator (UTM), and Latitude and Longitude representations amongst others

for users through a documented interface. This library can be used by equipment manufacturers without encumbrance, thereby easing adoption of common representations and reducing system integration effort over time. This library will not fit every need currently, it is a work in progress, and should evolve with features distinct from any one OS or middleware.

II. Previous Work

There is currently no standard set of data representation conventions in use globally across robotics work. There are many operating systems and middleware architectures, as detailed in RoSta[1], implementing various data representations, while some of them do not contain any representations nor conventions for robotics data natively.

There are many standards bodies and efforts, JAUS [2], BROS [3], NIST [4], [5], [6], CRAMOS [7], OMG [8] amongst others developing terminologies, ontologies, standard components, and data specifications in the hopes of greater interoperability. Work has continued for many years without the resolution of the needed single standard. In any case, if a single data standard specification proffered by one of these bodies was adopted by the research community majority, then that unified data standard specification would still need to be instantiated as a software library.

III. Approach

This section describes the general philosophy and the library internals in some detail. Given the global interest, it makes sense that a separate data standards library exists for use in any combination of operating system, middleware, or firmware to act as the bridge. As an alternative to the protracted standards development meeting process, DRDC is offering an open source data standards library, libdrdc, as a proposed solution that can be used immediately and further developed cooperatively by interested developers. This proposed solution is open to improvement and even adoption by standards bodies and equipment manufacturers. The only constraints on further development are to make certain that added functionality holds to the interface needs across scales and that these improvements are data-related. Standard functions should compute data values through the same logical steps in the hope of minimizing error and variation. The caveat is that there will remain some variation due to the various floating point unit (FPU) processor implementations (e.g. Motorola's 80 bit intermediate precision rounded to 64 bit for IEEE-754).

In order to ensure the widest possible adoption, this library was designed and built with GNU open source tools. This philosophy will allow adaptation and improvement without encumberance for anyone interested.

The library design philosophy adopts Marr's design principle of modularity [9], the UNIX library structure [10], Vaughn et. al. [11] GNU Autotools configuration system, and the guidance of Bruyninckx and Nilsson[12] to isolate and compartmentalize the functionality into a single stable subcomponent. A generic application can use the API without being a C/OOC/C++ application provided it can follow the proper function calling protocol. Figure 1 describes that the functionality at all levels is uniform and built up from the original C data structs and functions to maintain uniformity.

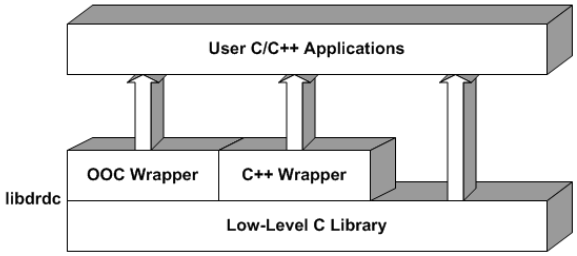


Fig. 1: libdrdc application progammer interface (API) description

The library consists of conversion functions and transformation functions. The former convert units of measure representations to or from the standard and the latter transformation functions compute homogenous transform matrices and vector-matrix operations. The default conventions are implemented and documented as the baseline; additional conversions or transformation can be included to the library source. Additional transformations and conversions are important and essentially necessary for any library attempting to act as a single point of truth[10] for data representations. So long as developers understand the libdrdc conventions and representations and convert personal values to standards before communication, then the data source will make sense to the data sink. If everyone uses this philosophy then portability of the data can be assured.

Low-Level C Library : The low-level C library provides a collections of small, efficient routines written in C for position, orientation, global coordinate systems, time, data and units of measure operations. It is ideal for small embedded microcontroller applications, and could be adopted to firmware builds using the low-level algorithms.

High-Level Wrapper Interface: The libdrdc high level wrapper encapsulates low level routines and data structures within more concise, robust, portable, and maintainable higher level object-oriented class interfaces (refer to Figure 1). The Object Oriented C (OOC) wrapper is designed for developing the complex embedded applications that use the C compiler. By adopting OOC methodology, libdrdc can use the C++ features such as data encapsulation and abstraction, access restriction and information hiding without a C++ compiler. The C++ wrapper provides a set of C++ wrapper classes that use the C library to make it easier to integrate

[1]http://wiki.robot-standards.org/index.php/Middleware

TABLE I: Global and Local Pose Data Structs

Data Struct Name	Size	Type	Frame
drdc_lPose_t	7-vector	Local	$\{\ell\}$
drdc_lePose_t	6-vector	Local	$\{\ell\}$
drdc_WPose_t	7-vector	Global	$\{W\}$
drdc_WUTMPose_t	7-vector	Global	$\{W-UTM\}$
drdc_WePose_t	6-vector	Global	$\{W\epsilon\}$
drdc_WeUTMPose_t	6-vector	Global	$\{W\epsilon-UTM\}$

into high-level applications.

Units of Measure: The library uses the System International (SI) units of measure for all internal representations and provides common conversions for mass, volume, et cetera from and to SI base and derived units. A complete list of base and derived units of measure are available in the website online documentation. If a researcher wishes to use another unit of measure then it is expected that all data must be converted into SI units before using the API functions and then from SI units into the same arbitrary units. If functions are not present to handle the conversions, then it is straightforward to include them.

libdrdc Unified Time Standard: There are many epochs and time offsets derived from the various system components like Global Position System (GPS) and Central Processor Unit (CPU) onboard clocks. This library uses Coordinated Universal Time (UTC) as the internal time standard and offers conversions to and from GPS, UTC, Unix, and RTEMS epochs' real-time clock settings are available. JAUS timestamp functions are included.

Global and Local Pose Conventions: The libdrdc library adopts a 6-vector and 7-vector approach for global and local pose representation using Cartesian or 2-dimensional spherical (Latitude, Longitude, Elevation) positions and Euler Angle (Euler-ZYX rotation sequence) or quaternion orientations. The library provides orientation and position primitives upon which the complete pose data is built. Table I describes the current uniform pose representations. Column 1 lists the data structures, column 2 describes the size of the vector, columns 3 and 4 list the type and frame internal representation. The libdrc library uses 2-dimensional spherical coordinates (latitude, longitude and elevation) for position and quaternion representation for orientation in a global pose $\{W\}$. Spherical coordinates are continuously differentiable and preferred over map projection positions like UTM for error reduction [13] especially when the position is far from the UTM zone origin. Global position should be kept in spherical coordinates and converted to Military Grid Reference System (MGRS) or UTM when necessary. Equations 1 through 4 describe the various world coordinate frame representations. A position 3-vector and a 4-vector quaternion for orientation represent the promoted global and local poses respectively. The world pose (7-vector) is defined in equation 1:

$$\vec{\mathbf{p}}_W \equiv [\varphi_W, \lambda_W, z_W, q_s, \vec{\mathbf{q}}]^T \tag{1}$$

where $\{W\}$ represents the global frame of reference, φ_W is the longitude displacement from the Prime Meridian in *radians*, λ_W represents the latitude displacement from the Equator in *radians,* and z_W represents the translation above or below Median Sea Level (MSL) in *metres*; positive values above MSL. A zero value for z_W represents MSL. The quaternion $\mathbf{q}$ defined by $(q_s, \vec{\mathbf{q}}) \equiv [q_s, < q_x, q_y, q_z >]^T$ is a dimensionless unit quaternion [14] where $\mathbf{q} \in \mathbb{R}$, $st\, q_{sxyz} \epsilon [-1, 1]$ and $q_s^2 + q_x^2 + q_y^2 + q_z^2 = 1$. Quaternions are prefered to Euler angles because they are singularity-free, continuously differentiable, do not suffer gimbal-lock, and many interpolations exist (quad, squad, et cetera.) to blend quaternions. The internal use of quaternions abstracts their complexity from the research developer. Refer to Erickson [15] for a proof that equates quaternion and Euler Angle representations[2].

Alternatively, libdrdc provides a global pose in UTM. The world pose (7-vector) is defined in equation 2:

$$\vec{\mathbf{p}}_{W-UTM} \equiv [x_W, y_W, z_W, q_s, \vec{\mathbf{q}}]^T \tag{2}$$

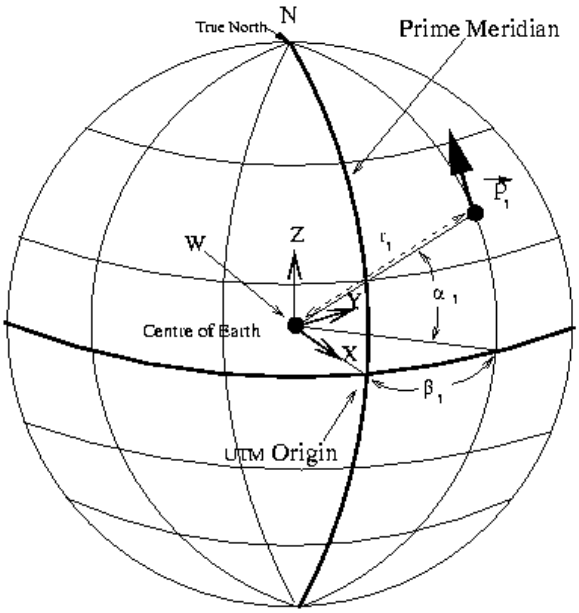
where $\{W-UTM\}$ represents the global frame of reference, and x_W and y_W are measured in *metres* and substituted for φ_W and λ_W respectively. The world coordinate reference frame $\{W-UTM\}$ requires additional information for proper conversion to and from spherical coordinates (Latitude zone, longitude zone, false easting convention, and geodetic datum) in the data struct **drdc_WPose_t**. This library uses World Geodetic System WGS-84 (the same as GPS) as the geodetic datum and the 500000m zone False Easting for all UTM pose representations. The library also provides representations for conversion from and to the world frame quaternions $\{W\}$ and $\{W-UTM\}$. These global frames are denoted $\{W\epsilon\}$ and $\{W\epsilon-UTM\}$(Equations 3 and 4) :

$$\vec{\mathbf{p}}_{W\epsilon} \equiv [\varphi_W, \lambda_W, z_W, \Psi, \Theta, \Phi]^T \tag{3}$$

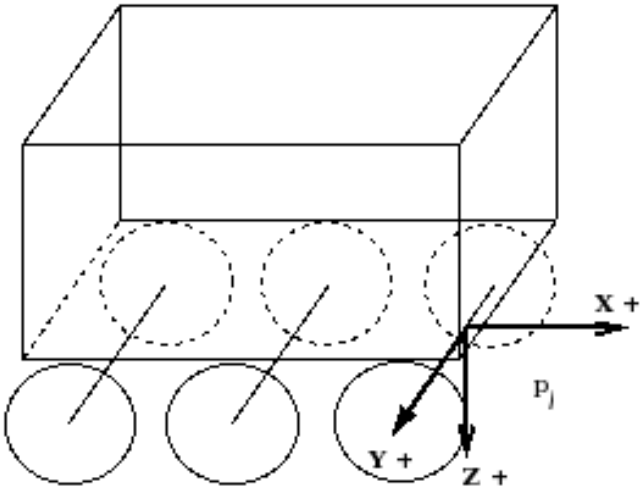
$$\vec{\mathbf{p}}_{W\epsilon-UTM} \equiv [x_W, y_W, z_W, \Psi, \Theta, \Phi]^T \tag{4}$$

The quaternion $q = [1.0, 0.0, 0.0, 0.0]$ is defined as the orientation origin and represents the System pointing True North, in the sense of Euler yaw, while having no pitch and no roll. Figure 2(a) illustrates vector $\overrightarrow{\mathbf{p}}$ is facing North along the global orientation origin towards the True North pole tangential to the geoid. Imagine a pencil facing True North and lying on a perfectly level desk as an analogy of the orientation origin in our representation. This orientation remains the origin no matter where on the Earth the pose is.

[2] *Publication available at http://cradpdf.drdc.gc.ca/PDFS/unc47/p525163.pdf*



(a) W World Coordinate Reference Frame



(b) Local Coordinate Reference Frame ℓ affixed to vehicle

Fig. 2:

The definition applies to the local robot coordinate frame of reference $\{\ell\}$. Equations 5 and 6 describe the local pose frame conventions. For interoperability, the robot origin reference frame should attach itself to the centre of the front axle or rotation axis, or single axis of wheels. If the robot has legs, then $\{\ell\}$ should attach itself to the bottom centre of the front hip joint. The exact placement must be visible from outside of the vehicle so as to allow for calibration through survey. Figure 2(b) demonstrates the local coordinate reference frame $\{\ell\}$ affixed to the front axle of a multi-axle UGV. If it is an aerial vehicle, the reference frame $\{\ell\}$ should attach to the bottom centre of the fuselage for fixed wing or rotary aircraft. Note in Figure 2 that the z-axis is downward from $\{\ell\}$. These standard frame conventions would help the machine vision systems to recognize robots and their estimated pose in the current field of view. This z-axis direction means that clockwise rotations are positive. This selection coincides with the *de facto* aerospace standard as well as the coordinate frame of reference for most commercial inertial measurement unit (IMU) sensors. The x-axis extends forward from the vehicle in the along-axis direction. This means that forward motion is positive when measured relative to a previous pose. The y-axis, or across-axis, extends out from the local pose origin in the right-handed sense.

$$\vec{\mathbf{p}}_\ell \equiv [x_\ell, y_\ell, z_\ell, q_s, \vec{\mathbf{q}}]^T \tag{5}$$

$$\vec{\mathbf{p}}_{\ell\epsilon} \equiv [x_\ell, y_\ell, z_\ell, \Psi, \Theta, \Phi]^T \tag{6}$$

ATLAS/BLAS Math Libraries Support : The BLAS (Basic Linear Algebra Subprograms) libraries has become a *de facto* standard for the elementary vector and matrix operations, where many open source scientific software projects like BOOST and GSL adopt the BLAS API at some level. This library provides the standard Fortran-77 BLAS interface to easily integrate BLAS library. Machine-optimized BLAS codes, when compiled by ATLAS, are typically faster than the compiled reference Fortran code by a factor between 2 to 10 times. Machine-optimized library should be used whenever it is available. If no such vendor supplied library is available, the supplied Fortran-to-C (F2C) converted routines from CLAPACK reference code can be compiled and used instead. The structure **drdc_HMatrix_t** provides the homogeneous transform matrix and **drdc_Vector_t** provides the generic vector base struct.

Documentation : Documentation for all functions and classes is compiled automatically by Doxygen into HTML and LaTeX formats. Documentation is part of the source tree and on the website for the current API interface features. The website contains the latest version of the generated documentation. In addition, an HTML dokuwiki located on the website describes the conventions as well as the interface description.

Time-scale Calculus: The library has an expanded extras section that has some time-scale calculus functions and it is planned to include more as time goes on.

Testing : This library has been extensively tested on different target systems to ensure its operation is correct. It uses the Geodetic Matlab Toolbox from Natural Resources Canada to generate expected test data for global coordinate systems conversions, Robotics Matlab Toolbox from CSIRO Division of Manufacturing Technology to test homogeneous, quaternion, roll pitch yaw (RPY), and manipulators transformations and the GNU units utility for standard unit conversions constants. All test suite elements are part of the libdrdc build.

Fig. 3: nScorpion Advanced EOD Robot

License: This software is licensed under the Lesser GNU Public License (LGPL v2.1) to promote standardization and the widest distribution and use and can be found at DRDC AISS website at `http://aiss.suffield.drdc-rddc.gc.ca/libdrdc/`.

IV. Results

The libdrdc data standards library has been successfully used on one prototype project. The nScorpion (Figure 3) is an advanced explosive ordnance disposal (EOD) robot built to extend the state of the art in EOD robotics. This robot employs 5 PowerPC MCU controllers and one Intel x86 CPU onboard communicating with the Intel x86_64 based control station via wireless connection.

V. Future Work

The current task is to expand and complete the time-scale calculus functions set. Once the library has survived a full coverage testing regime, extensions to the library will add internal state vector machinery for a common application approach. It is hoped that if the library is adopted by companies and labs in this field, that further processor implementations will be developed. A mid-term goal is the establishment of a gold standard server to remotely test computation results versus identical processor versions. This server would be available on the worldwide web to handle functions tests from anonymous sources. In addition to the current world pose conventions, an MGRS version is planned. This global MGRS pose should have $\{W - MGRS\}$ and $\{W\epsilon - MGRS\}$ transformations available.

VI. Conclusions

This paper presented the libdrdc data standards library. As an alternative to the protracted standards development meeting process, DRDC is offering an open source data standards library, libdrdc, as a proposed solution that can be used immediately and further developed cooperatively by interested developers. The libdrdc data standards library aims to be an open source multi-platform common tool outside any implementation with extensive routines designed to support distributed, embedded UGV applications at any scale.

VII. Acknowledgments

We would like to thank Dr. Bob Angus, Dr. Camille A. Boulet, Dr. Chris Weickert and Dr. Simon Monckton for their continued support of this project initiative.

References

[1] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Computing*, vol. 27, no. 1–2, pp. 3–35, 2001. [Online]. Available: citeseer.ist.psu.edu/whaley00automated.html

[2] JAUS WG, *Reference Architecture Specification: Message Set*. Washington DC: JAUS-AS-4 Working Group, August 2004, no. V3.2.

[3] K. Nilsson, T. Olsson, and H. Bruyninckx, "Basic Robotics Standards (BRoS)- Motivations and examples," in *Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, IEEE RAS/RSJ. IEEE Press, 2007.

[4] H.-M. Huang, K. Pavek, B. Novak, J. Albus, and E. Messina, "A Framework For Autonomy Levels For Unmanned Systems (ALFUS)," in *Proceedings of the AUVSI Unmanned Systems North America 2005*, AUVSI. Baltimore MD: AUVSI, June 2005, p. 9.

[5] H.-M. Huang, K. Pavek, J. Albus, and E. Messina, "Autonomy levels for unmanned systems (alfus) framework: An update," in *SPIE Defense and Security Symposium*, SPIE. Orlando FL: SPIE Press, April 2005.

[6] H.-M. Huang, *Autonomy Levels for Unmanned Systems (ALFUS) Framework Volume I: Terminology Version 1.1 (NISTSP 1011)*, 1st ed., ser. Special Publication, NIST, Ed. Gaithersburg MD: NIST, September 2004, vol. NISTSP, no. 1011.

[7] H. Bruyninckx, " Towards a Common Robotics, Automation and Manufacturing Operating System (CRAMOS)," November 2007.

[8] O. M. Group, "Request For Proposal Robot Technology Components (RTCs)," Object Management Group, First Needham Place 250 First Avenue, Suite 100 Needham, MA 02494, Request For Proposal ptc/2005-09-01, December 2005.

[9] D. Marr, *Vision*, 1st ed. W.H. Freeman and Company, 1982, p. 103.

[10] Eric Steven Raymond , *The Art of Unix Programming*, 1st ed. Addison-Wesley, 2003.

[11] G. Vaughn, B. Elliston, T. Tromey, and I. L. Taylor, *GNU AUTOCONF, AUTOMAKE, and LIBTOOL*, 1st ed. New Riders Publishing, 2001.

[12] H. Bruyninckx and K. Nilsson, "Design of (Robotics) Software Standards," November 2007.

[13] L. M. Bugayevskiy and J. P. Snyder, *Map Projections: A Reference Manual*. London UK: Taylor and Francis, 1995.

[14] J. Kuipers, *Quaternions and Rotation Sequences*, 5 ed. Princeton NJ: Princeton University Press, 1998.

[15] D. Erickson, "Standards for Representation in Autonomous Intelligent Systems: Promoting Interoperability amongst Autonomous Intelligent Systems," Defence R&D Canada – Suffield, 2, Technical Memorandum TM 2005-228, Dec 2005, DRDC-Suffield TM 2005-228.